EXTENSIBLE MARKUP LANGUAGE (XML) IN ELECTRONIC GOVERNMENT: SOME EXEMPLARY SCENARIOS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ayışığı B. Sevdik

January, 2004

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Varol Akman (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda Nur Alpaslan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Tuğrul Dayar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray

Director of the Institute

ABSTRACT

EXTENSIBLE MARKUP LANGUAGE (XML) IN ELECTRONIC GOVERNMENT: SOME EXEMPLARY SCENARIOS

Ayışığı B. Sevdik
M.S. in Computer Engineering
Supervisor: Prof. Dr. Varol Akman
January, 2004

In the last decade or two, information technology (IT) has evolved remarkably as can be perceived by the fact that the Internet has become an indispensable part of our lives. With the advent of the Internet in almost every aspect of our lives, we are said to be living in the information and knowledge age. However, there is one institution that is not making use of these developments in IT, despite the fact that it is the one which deals with information more regularly than any other. This institution that handles data and produces new information in every aspect of its work cycle is the government. Hence, applying the advances in information technology to the government, brings before us the concept of "*electronic government*" or as it is briefly called "*e-government*".

One of the evolving information technologies is the Extensible Markup Language (XML). The advent of XML has brought both a knowledge dimension and a service dimension to the universal information repository we call the Web, enabling platform-independent, machine-readable, structured data exchange. As it has quickly become the data exchange standard of the Web, the business and research communities are readily making use of XML and related Web services. Why not pick XML amongst the new information technologies to transfer the government into an e-government?

XML can be used to achieve interoperability in government-to-government services, a more citizen-centric approach and easier, platform-independent access in government-to-citizen services, and efficiency in government-to-business services; thus, transforming the traditional government into a more productive, paperless electronic government. In this research we explore the application of XML to the "e-government" concept. We present some exemplary e-government scenarios and show how XML can be used to implement them as we discuss the advantages and disadvantages of using XML in e-government.

Keywords: Extensible Markup Language (XML), Electronic Government (e-government), Web services.

ÖZET

ELEKTRONİK DEVLETTE GENİŞLETİLEBİLİR BİÇİMLEME DİLİ (XML) : BAZI ÖRNEK SENARYOLAR

Ayışığı B. Sevdik
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Varol Akman
Ocak, 2004

Son on ila yirmi yılda bilişim teknolojisi, Internet'in hayatımızın vazgeçilmez birer parçası olmasından da anlaşılabileceği gibi, muazzam bir şekilde gelişmiştir. Internet'in hayatımızın her alanına girmesiyle enformasyon ve bilgi çağında yaşadığımız söylenmektedir. Fakat bilgiyle diğerlerine göre daha düzenli şekilde uğraşmasına rağmen bir kurum var ki, bilişim teknolojilerindeki bu gelişmelerden faydalanmamaktadır. Çalışma çarkının her evresinde veri ile uğraşan ve yeni bilgi üreten bu kurum devlet kurumudur. Bilişim teknolojisindeki gelişmelerin devlete uygulanması karşımıza "elektronik devlet" ya da kısaca "e-devlet" kavramını çıkarmaktadır.

Gelişmekte olan bilişim teknolojilerinden biri de Genişletilebilir Biçimleme Dili (XML)'dir. XML'in ortaya çıkması platformdan-bağımsız, makine tarafından okunabilir, yapısal veri değişimi sağlayarak, Web adını verdiğimiz evrensel bilgi havuzuna hem bir bilgi boyutu, hem de servis boyutu getirmiştir. Hızlı bir şekilde Web'in veri değişim standardı haline gelmesiyle, iş ve araştırma çevreleri halihazırda XML'i ve bağıntılı Web servislerini kullanmaktadır. O halde devleti elektronik devlete dönüştürmek için neden yeni bilişim teknolojileri arasından XML seçilmesin?

XML, devletten-devlete hizmetlerde birlikte-işlerlik, devletten-vatandaşa hizmetlerde daha vatandaş-yanlısı bir bakış açısı ve daha kolay, platformdan-bağımsız erişim, ve devletten-iş dünyasına hizmetlerde verimlilik sağlamak için kullanılabilir. Böylelikle, geleneksel devleti daha üretken, kağıtsız elektronik devlete dönüştürür. Bu araştırmada XML'in "elektronik devlet" kavramına uygulanması incelenmektedir. Bazı örnek e-devlet senaryoları ve bunların uygulanmasında XML'in nasıl kullanılabileceği, XML'in e-devlette kullanımının avantaj ve dezavantajları tartışılarak sunulmaktadır.

Anahtar sözcükler: Genişletilebilir Biçimleme Dili (XML), Elektronik Devlet (e-devlet), Web servisleri.

Acknowledgements

I am deeply grateful to my supervisor Prof. Dr. Varol Akman, who has believed in me all the way, as well as provided his guidance and support.

I would like to address my special thanks to Prof. Dr. Özgür Ulusoy, Assoc. Prof. Dr. Ferda Nur Alpaslan, Assoc. Prof. Dr. Tuğrul Dayar, and Asst. Prof. Dr. Uğur Güdükbay for accepting to read and review this thesis and for their valuable comments.

I would also like to express gratitude to Rabia Nuray, who has provided me with her feedback, help and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the last decade or two, information technology (IT) has evolved remarkably as can be perceived by the fact that the Internet has become an indispensable part of our lives. With the advent of the Internet in almost every aspect of our lives, we are said to be living in the information and knowledge age. In this age, the way private sector businesses, government agencies, and other organizations communicate and exchange information among themselves and with the public is changing, as the Internet provides a universal link for data communication and exchange. Today with the advancement in IT, two different computer systems from opposite ends of the world can immediately exchange information through the Internet, in an unforeseeable way some twenty years ago. However, although information exchange has been so facilitated, it is not as easy to process this information obtained over the wire. This is where the base technology of the Web, known as the Hyper Text Markup Language (HTML), falls short and requires systems to use translation software in order to do more than merely view a disparate computer system's data.

Processing information obtained from a different system, involves making this data understandable by your own system. Using previously agreed-upon standards for labeling data, or tagging data, is one way of achieving machine-readability by disparate systems. The pioneer for tagging data is the Standard Generalized Markup Language (SGML) that was developed in the 1980's for structuring and publishing documents. Although SGML enables information reuse, it is a very complicated language and it is unsuitable for the universal information repository, we call the Web. HTML, the language that the Web is built upon, is a descendant of SGML developed for providing formatting on the Web. However, as previously pointed out, HTML data cannot be readily interpreted.

The Extensible Markup Language (XML) is one of the newly evolving information technologies and it makes up for the inadequacies of its ancestors, SGML and HTML. XML is a flexible, nonproprietary set of standards for tagging information so that it can be transmitted over a network such as the Internet and readily interpreted by disparate computer systems [27]. The advent of XML has brought both a knowledge dimension and a service dimension to the universal information repository we call the Web, enabling platform-independent, machine-readable, structured data exchange. The business and research communities are readily making use of XML and related Web services and it has quickly become the data exchange standard of the Web. However, there is one institution that is not making use of these developments in IT, despite the fact that it is the one which deals with information more regularly than any other. This institution that handles data and produces new information in every aspect of its work cycle is the government.

The application of advances in information technology to the government, to enable it to better perform its services and achieve its missions brings before us the concept of "*electronic government*" or as it's briefly called "*e-government*" or even "*e-gov*". Transferring the traditional government into an electronic government provides a more productive, efficient, citizen-centric, and paperless government. As its duties and

responsibilities involve interacting with a variety of entities, including citizens, private sector organizations, other governments and its own departments, it seems a technology enabling disparate systems to access and process each other's data would serve the government well. Thus, this research is motivated by the potential that XML has presented, through the capabilities it brought into the business and research communities, and the possibility of utilizing this potential in the fulfillment of the requirements of electronic government.

XML can potentially be used to achieve interoperability in government-to-government services, a more citizen-centric approach and easier, platform-independent access in government-to-citizen services, and efficiency in government-to-business services; thus, transforming the traditional government into a more productive, paperless electronic government. In this research we explore the application of XML to the "e-government" concept. We present some exemplary e-government scenarios and show how XML can be used to implement them as we discuss the advantages and disadvantages of using XML in e-government.

1.2 Organization of the Thesis

This thesis is organized as follows: The following chapter provides some background knowledge on the concepts of electronic government and Web services. It presents the requirements of e-government, as it describes the services an e-government is responsible to provide. Chapter 3 presents the Extensible Markup Language (XML) and its related standards, including Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI). The chapter provides thorough explanation of each standard and gives examples for clarification where appropriate.

Chapter 4 discusses the adoption of XML and the related standards presented in the previous chapter by the electronic government. Both the advantages and disadvantages

of using XML in the e-government environment are explored. The next chapter follows up on the findings of Chapter 4, as it presents possible scenarios that may take place in an electronic government. The scenarios are explained in detail as implementation possibilities through XML are investigated. Finally, Chapter 6 summarizes the work completed in this research and presents some concluding remarks, as well as a discussion of possible future work.

Chapter 2

Background

2.1 Electronic Government

In the last decade or two, information technology (IT) has evolved remarkably as can be perceived by the fact that the Internet has become an indispensable part of our lives. With the advent of the Internet in almost every aspect of our lives, we are said to be living in the information and knowledge age. And in this information and knowledge age, there is no institution that produces raw data and new information with more regularity than the government. Furthermore, we can say that information is at the heart of every policy decision, response, activity, initiative, interaction and transaction between government and citizens, government and businesses, and among governments themselves [1]. This leads to the idea of applying the advances in information technology to the government, bringing before us the concept of "*electronic government*" or as it's briefly called "*e-government*" or even "*e-gov*".

What makes a government an "e-government" and how do we apply the new information technology practices to the institution of government to provide e-government? More importantly why do we want an electronic government? Is it simply because it's the latest fashion all around, or does it have its benefits?

2.1.1 What Makes a Government an "e-Government"?

As stated in the Turkish Informatics Council e-Government Working Group Report of 2002 [2], "e-Government, in its most simplest form, is defined as; 'the uninterrupted and secure, mutual execution of the duties and services the government is responsible to perform towards citizens and the duties and services of citizens towards the government, in electronic communication and processing environments.'" So, the key in making a government implement the "*e*" concept is in enabling it to perform its services and achieve its missions in electronic communication and processing environments; i.e. depending upon information technology (IT). Hence, a government must apply IT to all its basic services and units from citizens to both public and private organizations. These basic services that an e-government must provide can be classified as 'government-to-citizen', 'government-to-business' and 'government-to-government'.

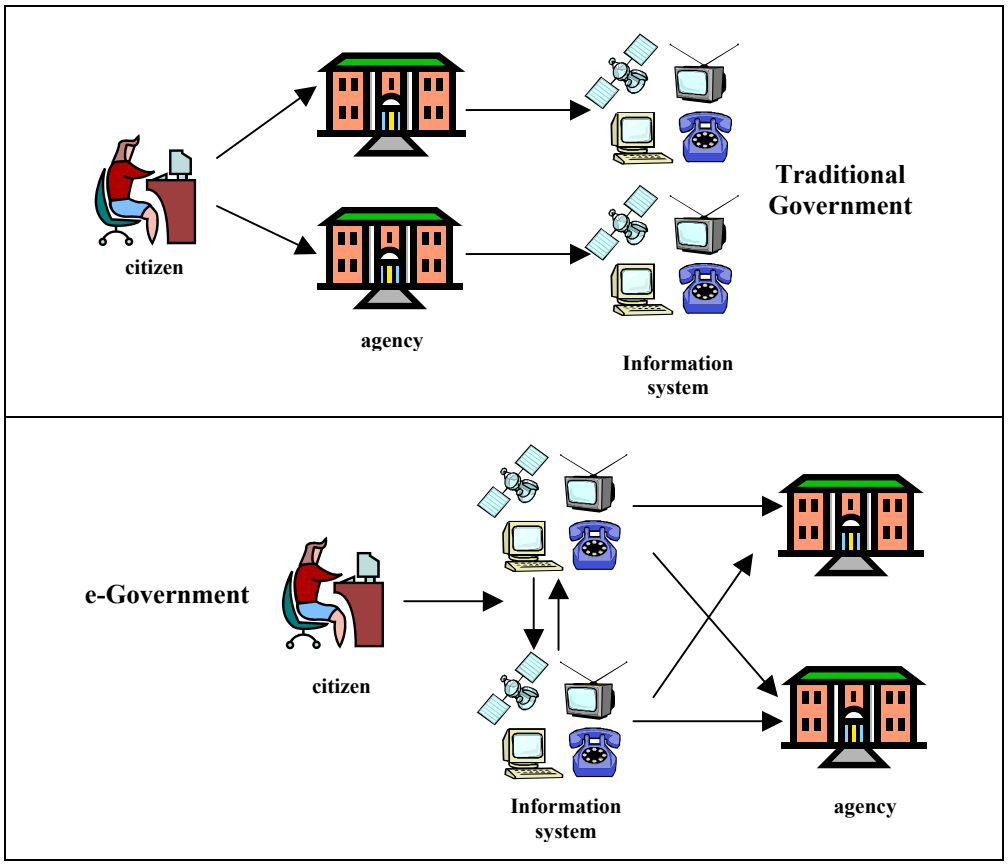


Figure 2.1: Traditional Government vs. e-Government

2.1.1.1 "Government-to-Citizen" Services

Perhaps the most important and best known services the government performs are the ones it provides to us, to its citizens. Traditionally, these services are provided to citizens through long hours in line, in front of government offices, where one has to traverse several of, in order to complete a single transaction/process. An e-government offers these services in an electronic environment where waiting in line is not necessary.

The most popular method for e-governments to provide their government to citizen services is through the use of a government portal. A portal enables a citizen to access a variety of services from a single point (pay a parking fine, renew professional license, pay taxes, etc.), instead of having to traverse many different (numerous) departmental Web sites in order to access the same services. A government portal must be informative, as well as provide governmental services for the citizen, with easy search and processing capabilities, through single point-entry. It should aggregate services across departments and organize its services in a citizen-centric way presenting a single face to the citizen. In this way, the portal should save the citizen from the burden of having to know which department(s) to go to for a desired service. The portals should also communicate with and make use of citizen services provided by private organizations and should not be restricted to services of public organizations. In an ideal (seamless) e-government scenario, the government portal should present the government in a simple way, hiding organizational complexities from the citizens, enabling them to handle personal transactions interactively on-line. Here a transaction may involve the communication of two or more government organizations, or a government organization with a private sector organization, in the background where it is invisible to the citizen. An exemplary scenario in a seamless e-government portal implementation can be given as follows:

Let's say you bought a new house and you want to get a title deed. From the e-government portal, you click on the "house/real estate" button and a registration/application form along with house insurance prices of different companies

and reminders related to real estate, etc. appear. Here, you can apply for your title deed, pay for insurance or pay your real estate tax. If a financial transaction is required, the related form will have secure credit card payment capability. Once you have completed your registration procedure, the portal will send messages through e-mail, fax, GSM, etc. to remind you of upcoming tax or insurance payments pertaining to the real estate, at appropriate times.

2.1.1.2 "Government-to-Government" Services

In order to provide services to citizens as in the scenario in the previous section, although invisible to the citizen, the government has to maintain a secure and flawless information flow between its own agencies and departments. Data has to be transferred and updated to or from disparate legacy systems and databases of the responsible departments so that the requested transaction can be processed. This is what we have previously classified as 'government to government' services. Constructing the e-government, also involves applying the '*e*' concept to these internal services; thus having e-agencies or e-departments.

Establishing e-agencies and e-departments involves government employees to be connected to each other using intranet technologies, where they have different authorization levels that typically require IDs and passwords. Intranets allow government employees of a certain agency to collaborate both within their own agency and across other agencies, by providing secure electronic workflow and data exchange. This structure in turn, allows government agencies to provide their services in the most efficient and effective way.

2.1.1.3 "Government-to-Business" Services

The last basic group of services the government performs is the services it provides to the business sector; hence they can be called "government-to-business" services. These

include both the services private sector organizations request from the government and the services the government requires from business organizations.

First, let's consider the services government has to provide to businesses. A business needs to get a permit or a license, pay sales taxes or customs taxes, etc. The traditional government architecture sets the perception that the government (bureaucratic) work forms a bottleneck for the private sector in that, a business cannot get its work done as quickly when interfacing with the government as it can when interfacing with other private sector organizations. It takes too much time for a business organization to get a permit, to renew a license, to pay taxes, etc. since these interactions with the government are paper-based. In the information age, almost all businesses have an on-line appearance, take part in e-commerce and use every aspect of information technology to perform their work in a more efficient and fast manner. However, in their interactions with the government they cannot take advantage of the same benefits. This is a traditional government's interaction with businesses. An e-government, on the other hand, would be eliminating the long waits and inconvenience, since it makes use of the latest information technologies when responding to the private sector.

A most striking example of governments services to business and how an e-government provides these services was mentioned in the 'Turkish Informatics Council e-Government Working Group Report' of 2002 [2]. According to the report, every year 2.5 million trucks apply to the Czech customs and this number increases from year to year. 9000 customs officers are working in the Czech customs, registering every good entering the country, accumulating taxes and statistical data, and performing other customs procedures. It has been calculated that on average, a truck driver is kept waiting for 12 hours to get through customs. This was a very inconvenient situation for all the truck drivers, the businesses, and the customs officers. The Czech authorities finally found the solution to the continuous increase of traffic by putting the customs operations on-line. For this purpose, with 75 of them at the frontier, total of 200 customs officers have been connected to each other via the Internet, and at the beginning of the year 2000, 80% of customs declarations have been sent over the Internet. Thus, the collection

of customs tax has been fully automated as a step towards becoming an e-government, freeing the truck drivers from the long waiting hours and speeding up the work of businesses.

In the same way, the government may require private sectors services such as buying supplies. An e-government uses on-line trading systems to fulfill these requirements. Thus, in an e-government, paper-based government-to-business operations are replaced by linking external stakeholders (such as title companies, insurance companies, suppliers, and attorneys) to the government through an extranet, allowing fast and efficient completion of the mutual operations.

2.1.2 Why e-Government?

The previous section talked about how to establish an e-government. However, what is more important is why e-government is such a desired goal. What is expected to be established by transforming the government into an e-government? What are the benefits of an e-government?

The goals of e-government can be listed as below:

- transparency of the government,
- fast and efficient execution of the government,
- increased citizen participation in governance at every level,
- prevention of work and data recursion by inter-organizational data exchange,
- making life easy for citizens that the government serves, and
- improved and fast decision-making process based on information by authorities.

The implications arising from the achievement of these goals of e-government are as follows:

- transformation of the government like no previous reform or reinvention initiative,
- time gain,
- lower costs and higher efficiency,
- increased contentment,
- improved economic growth,
- improved quality of life,
- increased individual participation,
- decreased paper-dependency and use,
- minimization of human errors by single-point access to accurate information necessary for the public services to citizens,
- Lower costs in the long run, although the initial costs of launching information technologies is high,
- faster service provision,
- lower administration costs,
- access to accurate information, etc.
- also, e-government will provide ease and speed in the decision-making process for both the citizen and the public,
- more citizen-centric approach empowering individual citizens,
- improved relations with citizens and the establishment of trust,
- make government and its services more accessible,
- facilitate social inclusion, and

- use government resources effectively and efficiently.

2.1.3 Requirements for Establishing an e-Government

There are some requirements a government has to fulfill in order to become a well-established e-government. When implementing the e-government concept, pitfalls such as duplicative efforts, failure to consider infrastructure requirements, and implementing technologies that are not sufficiently flexible or scaleable should be avoided. In order to be a more citizen-centered, customer-focused government that maximizes technology investments, e-government should consist of initiatives that are transformational in nature and offer the opportunity to simplify and unify processes used by the government [27]. Following is a list of requirements or considerations for establishing e-government:

- leverage information technology investments,
- avoid unnecessary duplication of infrastructure and major components,
- link business processes through shared, yet sufficiently protected information systems,
- leverage disparate business processes, services and activities that are located outside a given governmental agency's boundaries,
- avoid implementation technologies that are not sufficiently flexible or scaleable,
- consider infrastructure requirements,
- avoid duplicative efforts,
- eliminate use of proprietary software dependencies,
- adopt voluntary industry standards maintained by group consensus,
- try to achieve data standardization, including a common vocabulary and data definition,
- ensure data consistency,

- make the principle “enter once, use often” applicable,
- appropriate security planning and monitoring must be done to prevent unauthorized access to government information,
- when dealing with highly sensitive information, on top of establishing a ‘trusted network’, each electronic record must be secured individually from inappropriate access or change,
- standardize common functions and customers in order to implement changes more quickly,
- allow a wide range of users and access methods (personal contact, electronic, paper, service providers) for accessing public information, and
- properly secure the privacy information maintained to protect the privacy of the citizen.

2.2 Web Services

In the new millennium, we have discovered the power of the Internet to connect every home, every person, everywhere that it’s available and we have made the Web an important part of our lives. The Web, due to its simplicity and ubiquity, forms a platform, which demolishes geographical constraints and where anyone who is on-line can access an immense amount of information as well as exchange information with any other party. Although the Web started out to be used solely for its capabilities to make information available (as an information distributor), today it has also become a platform for providing services. Today, most businesses have a presence on the Web, providing us their services on-line. Services enabling us to perform banking transactions, buy flowers or a car, or even do our grocery shopping, simply by going on-line. Not by downloading special software, or buying specialized equipment, but simply by owning a computer and an Internet connection we can take advantage of such services. All these capabilities are made possible by the use of Web services.

Web services are component services that can be reused to build larger services. For example a credit card transaction function can be reused by many different companies offering business services ranging from selling opera tickets to accepting school tuitions. They "are a new breed of Web application," as the IBM's web services tutorial [3] describes. "They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes…"

What is so wonderful about Web services and why have they caught up so much attention? The key that has enabled Web services to catch up so much attention lies in the accessibility and widespread use of the Web bringing about the possibility of interoperability – a property not quite available in a traditional middleware environment.

The basic Web services platform is XML (Extensible Markup Language) over HTTP (Hyper-Text Transfer Protocol). XML is a meta-language in which specialized languages marking data to make it machine-readable can be written. Hence, through which software components can interact. In the case of Web services, XML is used to define the public interfaces and bindings that take place in the interactions between clients and services, or between service components. Basically what happens is that an XML message gets converted into a middleware request, the business logic performs the request and the results are converted back into an XML response which is then transferred over HTTP to the originator of the request.

The Web services platform, apart from XML and HTTP, requires some support services for discover, transactions, security, authentication, etc. The most important platform support services are SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery, and Integration), and WSDL (Web Services Description Language). These are also known as XML Web services standards.

Web services are based on the principle of openness. Hence, it's this foundation that separates the Web and the web services movement from others. All the standards, services and technologies mentioned up to this point are open and cross-platform standards.

Chapter 3

XML and Related Technical Standards

3.1 Extensible Markup Language (XML)

Extensible Markup Language, abbreviated as XML, is a meta-language designed to structure data in a meaningful way, which is machine-readable, extensible, and platform-independent. Its foundations lay in the Standard Generalized Markup Language (SGML), which is a more complicated language enabling structuring of documents and information reuse; but which is also difficult to learn, difficult to use in the Web environment, and mostly just used for technical documentation. XML was intended to be more regular, simpler and more suitable for Web use than SGML. Inheriting the best parts of SGML and benefiting from the HTML experience, XML began its development in 1996 by the XML Working Group, which was known as the SGML Editorial Review Board at that time, of the W3C. The language became a W3C Recommendation on February 10th, 1998 as XML 1.0. This initial recommendation was revised and obtained its final form as XML 1.0 (second edition) in October 2000.

Although both are markup languages used on the Web, there is one important difference between XML and HTML; that is, XML provides meaningful (descriptive) tags, or provides information about its content making it readable by both humans and machines, whereas HTML solely provides layout with a fixed set of tags. Figure 3.1

below displays a sample citizen record (a) as an HTML document and (b) as an XML document.

(a)

```
<html>
<body>
<p>
<br>Sevdik Ayisigi Basak</br>
<br>1981-01-01</br>
<br>1234567890</br>
</p>
</body>
</html>
```

(b)

```
<?xml version="1.0" encoding="UTF-8"?>
<Citizen>
  <Name>
     <LastName>Sevdik</LastName>
     <FirstName>Ayisigi Basak</FirstName>
  </Name>
  <DateofBirth>1981-01-01</DateofBirth>
  <SocialSecurityNumber>1234567890</SocialSecurityNumber>
  <MaritalStatus Single="true"/>
</Citizen>
```

Figure 3.1: A sample citizen record (a) as an HTML document and (b) as an XML document

XML makes use of tags to structure data, but it does not specify how each tag should be interpreted. It provides user-defined, processing-independent markup. The tags in an XML document specify structures known as *element*s. The *element*s in the XML document of Figure 1.b are `Citizen`, `Name`, `LastName`, `FirstName`, `DateofBirth`, `SocialSecurityNumber`, and `MaritalStatus`. *Element*s may contain data between their start- and end-tags, have other elements as children, and have *attributes* that provide additional information about the *element*. An *attribute* appears in the start-tag of an *element* and has the form `attributeName="attributeValue"`. An example is seen in Figure 1.b, where the `MaritalStatus` *element* has the *attribute* `Single` with a value of "true". Here, the `MaritalStatus` *element* is an empty *element* because it does not contain any data and unlike the other *element*s in the figure, is represented by a

single empty *element* tag. Both a start-tag and an end-tag delimit all the other *element*s, as they are non-empty. Another property of *element*s is that they can contain other *element*s as children within themselves. The top-level *element* of an XML document is the *root element*, which in the case of Figure 1.b is the `Citizen` *element*.

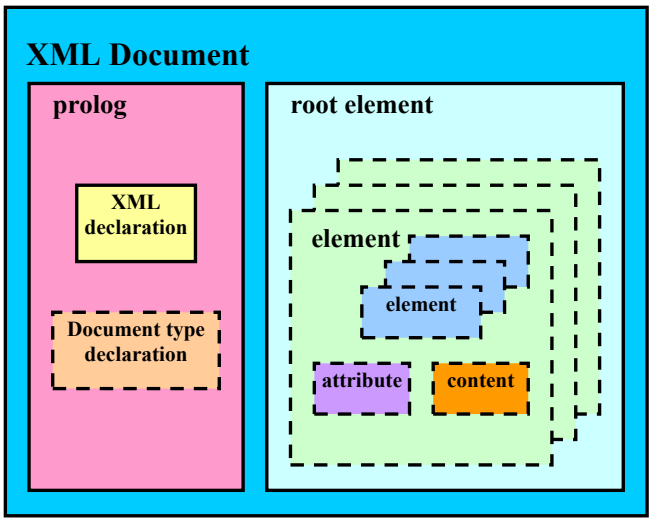


Figure 3.2: Structure of a well-formed XML document

An XML document that conforms to all the XML syntax rules is called a *well-formed* XML document. A *well-formed* document must start with a *prolog* and have only one *root element*. The *prolog* of an XML document contains an *XML declaration* and a *document type declaration*, where an *XML declaration* contains information like the XML version used and the *document type declaration* information providing a grammar for the document. Also, a *well-formed* XML document's all non-empty *element*s must be delimited by start- and end-tags and all its empty *element*s must have the empty tag syntax. Another obligation is that, all of the *element*s of a *well-formed* XML document must nest properly within each other, meaning that if the start-tag of an *element* is in the content of another *element*, its end-tag must also be in the content of the same *element*. Figure 3.2 shows how a *well-formed* XML document is structured. It should be noted that the XML document in Figure 1.b is also a *well-formed* document.

The grammar contained or pointed to by the *document type declaration* of an XML document is known as a *document type definition* (DTD). DTDs define the structure of XML documents as they allow the specification of the set of tags, the order of tags, and the associated *attribute*s. A DTD can either be included in the XML document or reside in a separate file that the XML document contains a reference to. If an XML document is *well-formed* and at the same time conforms to the DTD specified in its *document type declaration*, then it is called a *valid* XML document.

Up to this point, we have explained the basic structure of XML and XML 1.0. Actually, there is a whole family of XML technologies beyond the XML 1.0 specification. "The XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks [4]. The following subsections will present the most important members of this family.

3.1.1 XML Namespace

As previously stated, XML tags are user-defined and anyone can define their own set of tags to describe their data. Also, XML allows reusing or combining formats to define a new formats. Thus, this brings up the question of differantiating between two *element*s with the same name belonging to different formats when combining these formats. In such a case, name confusion can be eliminated by the *XML namespace* mechanism. An *XML namespace* is a collection of names, identified by a URI reference, which are used in XML documents as *element* types and *attribute* names [5]. *Element* tag names in a particular *namespace* must be unique. Referring back to Figure 1.b, if we take into consideration the `name` *element*, it can be easily understood from the document that this *element* refers to the name of a citizen. However, we might have another document to keep a list of departments which also contains a `name` *element* referring to the name of a department. In a document which utilizes these two formats, we would use separate *namespace*s as in Figure 3.3. Here, two *namespace*s are used by defining the *namespace* prefixes `citizen` and `department`. It should be noted that if an *element* is associated with a *namespace*, then all its children are also associated with the same *namespace*.

```
<?xml version="1.0"?>
<CitizenList
   xmlns:citizen="http://www.trafficDept.gov/citizenRecords/"
   xmlns:department="http://www.DirectorateofSecurity.gov/department/"
>
... <citizen:Name><LastName>Sevdik</LastName> ... </citizen:Name>
... <department:Name>Traffic Department</department:Name> ...
```

Figure 3.3: Example of namespace use

3.1.2 XML Schema

Not all XML documents are structured using a DTD. The expressive power of DTDs seems limited and they are not in XML. Thus, *XML Schema* was developed by the W3C and the XML Schema W3C Recommendation was published in May 2001 [6] to provide better functionality than DTDs. *XML Schema*s allow users to define their own structures using XML syntax. They are XML documents providing a means for defining the structure, content and semantics of other XML documents. In addition to the datatypes supported by DTDs, *XML Schema*s support other datatypes such as *string*, *date*, *URL*, etc. *XML Schema*s have more expressive power than DTDs as they are developed to make up for the lacking properties of DTDs. They are also extensible, as they allow the combination of schemas to produce a new one. Thus, *XML Schema*s support the modularity principle of XML at the structure definition level.

3.1.3 The Extensible Stylesheet Family (XSL)

One of the most important properties of XML is its property to separate content and presentation. Separating the document's content and the document's styling information allows displaying the same document on different media (like screen, paper, cell phone), and it also enables users to view the document according to their preferences and abilities, just by modifying the styling specification [7]. This is established with the use

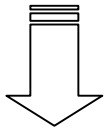
of another family within the XML family, known as the *Extensible Stylesheet Family (XSL)*. XSL is a family of recommendations for defining XML document transformation and presentation. [8] It is made up of three parts: The *XSL Transformations (XSLT)*, which is a language for transforming XML into other formats; the *XML Path Language (XPath)*, which is a language enabling reference or access to parts of an XML document; and the *XSL Formatting Objects (XSL-FO)*, which enables specification of formatting semantics.

3.1.3.1 XSL Transformations (XSLT)

XSL Transformations (XSLT) is an XML-based language for transforming XML documents into other text formats. XSLT Version 1.0 was published as a W3C Recommendation in November 1999 [9]. The most common use of XSLT today is for transforming one type of XML document into another type of XML document, which helps alleviate schema incompatibilities [10].

Figure 3.4 below shows a transformation between different versions of citizen documents. Such a transformation can be performed using XSLT.

```
<v1:ctzn xmlns:v1='urn:citizen:v1'>
   <FirstName>Ayisigi Basak</FirstName>
   <LastName>Sevdik</LastName>
   <DateofBirth>1981-01-01</DateofBirth>
   <SocialSecurityNumber>1234567890</SocialSecurityNumber>
   <MaritalStatus Single="true"/>
</v1:ctzn>
```



```
<v2:citizen xmlns:v2='urn:citizen:v2'>
   <Name>Ayisigi Basak Sevdik</Name>
   <SSN>1234567890</SSN>
</v2:citizen>
```

Figure 3.4: Transformation between different versions of citizen records

An XSLT transformation is specified as a *well-formed* XML document that conforms to the *XML Namespace*s. These may include both *element*s that are defined by XSLT and *element*s that are not defined by XSLT. XSLT-defined *element*s are distinguished by belonging to the *XSLT namespace*. Such a transformation in XSLT is called a *stylesheet* and it describes rules for transforming a source tree into a result tree. A stylesheet contains a set of template rules consisting of a *pattern* and a *template*. The *pattern* is matched against nodes in the source tree, and the *template* is instantiated to form part of the result tree. Thus, a stylesheet can be applicable to many documents that have similar source tree structures.

XSLT transformations can be exemplar-based, procedural, or declarative. Exemplar-based transformations allow production of dynamic content at appropriate locations of an XML document by inserting XSLT programming constructs. An example of an exemplar-based transformation is given in Figure 3.5, which performs the transformation of Figure 3.4.

```
<!—exemplar document -->
<v2:citizen
   xmlns:v1='urn:citizen:v1'
   xmlns:v2='urn:citizen:v2'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
   xsl:version='1.0'>
   <Name><xsl:value-of select="concat(/v1:ctzn/FirstName,' ',
   /v1:ctzn:LastName)"/></Name>
   <SSN><xsl:value-of
   select='/v1:ctzn/SocialSecurityNumber'/></SSN>
</v2:citizen>
```

Figure 3.5: Example of an exemplar-based XSLT transformation

XSLT makes its transformation logic reusable by making use of *template*s. *Template*s can be called like functions to output a portion of the resulting document. This type of transformation is a procedural transformation. Figure 3.6 gives an example of such a

transformation, which defines *template*s for the sample transformation defined in Figure 3.4.

```
   <xsl:transform
      xmlns:v1='urn:citizen:v1'
      xmlns:v2='urn:citizen:v2'
      xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
      version='1.0'>

      <!—outputs name element -->
      <xsl:template name="outputName">
         <Name><xsl:value-of        select="concat(v1:ctzn/FirstName,'
         ',v1:ctzn:LastName)"/></Name>
      </xsl:template>

      <!—outputs SSN element -->
      <xsl:template name="outputSSN">
         <SSN><xsl:value-of select='v1:ctzn/SocialSecurityNumber'/></
         SSN>
      </xsl:template>

      <!—root template: main entry point -->
      <xsl:template match="/">
         <v2:citizen>
            <xsl:call-template name="outputName"/>
            <xsl:call-template name="outputSSN"/>
         </v2:citizen>
      </xsl:template>
</xsl:transform>
```

Figure 3.6:Example of a procedural XSLT transformation

XSLT transformations can also be defined using a declarative model. This is a more powerful and flexible model as it is based on associating *template*s with *pattern*s relative to the input document. This model enables partitioning of transformation logic into modules that are automatically associated with a portion of the input document. Thus, a given *template* is just associated with a particular portion and the time and method of calling the *template* is left to the processor. Figure 3.7 displays an example of such a

declarative transformation, again performing the citizen records version transformation of Figure 3.4.

```
<xsl:transform
   xmlns:v1='urn:citizen:v1'
   xmlns:v2='urn:citizen:v2'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
   xsl:version='1.0'>

   <!—override built-in template for text/attributes -->
   <xsl:template match="text()@*"/>

   <!—template for SocialSecurityNumber elements -->
   <xsl:template match="SocialSecurityNumber">
      <SSN><xsl:value-of select='.'/></SSN>
   </xsl:template>

   <!—template for FirstName elements -->
   <xsl:template match="FirstName">
      <Name><xsl:value-of select="concat(.,' ',following-sibling::
        LastName)"/></Name>
   </xsl:template>

   <!—template for v1:ctzn elements -->
   <xsl:template match="v1:ctzn">
      <v2:citizen>
         <xsl:apply-templates select="*"/>
      </v2:citizen>
   </xsl:template>
</xsl:transform>
```

Figure 3.7: Example of a declarative XSLT transformation

Declarative and procedural transformations enable outputting the result document in XML, HTML, or straight text; however, exemplar-based transformations only allow XML output. The former two models also allow partitioning of transformations into multiple source files.

3.1.3.2 XML Path Language (XPath)

The *XML Path Language (XPath)* is a language for addressing, or decribing locations in XML documents. XPath is used in XSLT stylesheets to specify the portions of the XML document that are to be transformed. It is also utilized by XML Pointer (XPointer), which will be described in the following subsections. XPath was developed to provide common syntax and semantics between XSLT and XPointer, and was published as a W3C Recommendation in November 1999 [11].

In order to perform addressing of an XML document, XPath provides basic facilities for manipulation of *string*s, *number*s and *boolean*s. It uses a compact, non-XML syntax to make it easy to use within URIs and XML *attribute* values. In addition to addressing, XSLT uses XPath for *pattern* matching.

The most important construct of the XPath syntax is the location path. They are special syntaxes defined to specify certain locations of an XML document. An example is seen in Figure 3.7 as `following-sibling::LastName` that selects the next sibling of the context node, which in the example is the `Name` *element*, as `LastName`.

3.1.3.3 XSL Formatting Objects (XSL-FO)

The Extensible Stylesheet Language (XSL) defines a set of formatting objects that describe how a data should be formatted. In order to distinguish it from XSLT, it is referred to as *XSL Formatting Objects (XSL-FO)*.

Formatting interprets the result tree in its formatting object tree form to produce the presentation intended by the designer of the stylesheet from which the XML *element* and *attribute* tree in the "fo" *namespace* was constructed. Semantically, each formatting object represents a specification for a part of the pagination, layout, and styling

information that will be applied to the content of that formatting object as a result of formatting the whole result tree. Each formatting object class represents a particular kind of formatting behavior.

Unlike the case of HTML, *element* names in XML have no intrinsic presentation semantics. Absent a stylesheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL-FO provides a comprehensive model and a vocabulary for writing such stylesheets using XML syntax [12].

3.1.4 XML Linking (XLink)

XML Linking (XLink) provides linking capabilities to an XML document by describing a way to add hyperlinks to an XML file. XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to assert linking relationships among more than two resources, associate metadata with a link, or express links that reside in a location separate from the linked resources [13].

An example of an XLink is displayed in Figure 3.8. Here, the similarity of XLinks to HTML links can be perceived; however, unlike HTML links XLinks enable the definition of the behavior of a link.

```
<STUDENT xmlns:xlink=http://www.w3.org/TR/xlink
   xlink:type="simple"
   xlink:href="http://www.bilkent.edu.tr/~ayisigi/"
   xlink:role="ayisigi_sevdik_homepage"
   xlink:show="embed"
   xlink:actuate="onLoad">
Ayisigi B. Sevdik</STUDENT>
```

Figure 3.8: A sample XLink

The XLink specified in Figure 3.8 has the *attribute*s *href*, *role*, *show* and *actuate*. The first one (*href*) enables the specification of a URI, and *role* indicates the purpose of the

linked *element*. The *attribute show* specifies the action to be taken when the linked *element* is loaded, and *actuate* specifies when this action should take place.

3.1.5 XML Pointer (XPointer)

XML Pointer Language (XPointer) is a language which uses XPath to reference other resources. The XPointer specification [14] has been divided into XPointer Framework [15], and three additional schemes: the *XPointer element()* scheme [16], *XPointer xmlns()* scheme [17], and the *XPointer xpointer()* scheme [18]. XPointer is built on top of XPath and provides extensions to add the ability to identify locations that are not single, whole *element*s (such as those corresponding to typical selections and selection points in some user interfaces), and to combine string matching with the other location methods provided.

3.1.6 Properties of XML and Current Use

The most significant properties of XML are its modularity, processing-independence, the fact that it is both machine and human readable, it is user-defined, and perhaps its most important property, the ability to provide semantics. XML is a text format and hence, criticism arises due to the implication of this property of XML documents being larger, wordy files. However, with the help of compression programs, this property should not create any problems. Last, but not least, the features that will enable XML's widespread adoption are that it is license-free, well-supported, and platform-independent. Thus, because it brings along such strong properties, despite the fact that XML Recommendations are relatively new specifications, there are already millions of XML pages appearing on the Web. Virtually all application domains are looking to use XML to define and exchange structured information [19].

3.2 Simple Object Access Protocol (SOAP)

SOAP is an XML messaging specification. Since it's XML based, SOAP is neither hardware, platform, nor programming language dependent.

DevelopMentor, Microsoft, and UserLand Software submitted SOAP to the IETF (The Internet Engineering Task Force) as an Internet public draft in December 1999 and there was SOAP 1.0. On the year 2000, the W3C (the World Wide Web Consortium) formed the XML Protocol Working Group and started to work on SOAP 1.1. Since then, W3C and other interested parties have revised it, establishing its last and final form: SOAP Version 1.2, which was published as a W3C recommendation on June 24th, 2003 [20].

The editor's draft [21] of the W3C recommendation defines SOAP Version 1.2 as "a lightweight protocol for exchange of information in a decentralized, distributed environment." The recommendation consists of three parts. Part 0 is a non-normative document explaining the features of the SOAP Version 1.2 specifications. Part 1 of the recommendation describes the messaging framework, Part 2 specifies a set of adjuncts that may be used with Part 1; namely SOAP data model, SOAP encoding, and SOAP RPC representation, the SOAP HTTP binding, and such. The W3C XML Protocol Working Group also tracked implementations [22] of SOAP 1.2 from seven different W3C member organizations: Apache, BEA, Microsoft, SOAPLite, Systinet, TIBCO, and White Mesa. This was done in order to prove and improve the interoperability of SOAP 1.2. Apart from the organizations mentioned, other major vendors including IBM, Oracle, and Sun Microsystems, are supporting the SOAP specification.

SOAP basically allows methods to be invoked against endpoints over HTTP or any other underlying protocol. Upon receiving a request from the client-side, the server-side software is expected to execute some code that corresponds to the requested service. The server-side may be running on any platform and may be implemented in a variety of ways. Some possible reactions to a request are:

- a CGI program may run,
- a COM (Component Object Model) may run,
- a Perl script may be activated,
- an Apache module may be called,
- a Java Servlet or ISAPI extension may be invoked,
- an ASP or JSP page may be called,
- an XSLT may be run against the request,
- a servant may be dispatched inside a CORBA ORB, or
- a human may read the request and start typing a response.

This displays the interoperability property of SOAP, which enables it to run on disparate platforms using different implementation technologies.

SOAP 1.2 specification describes data formats, and the rules for generating, exchanging, and processing messages using those formats. The specification gives the basics of SOAP, but provided that they conform to mandatory requirements of the specification extended implementations may be built. SOAP has been designed to have a relatively small number of dependencies on other XML specifications, none of which are perceived as having prohibitive processing requirements [20].

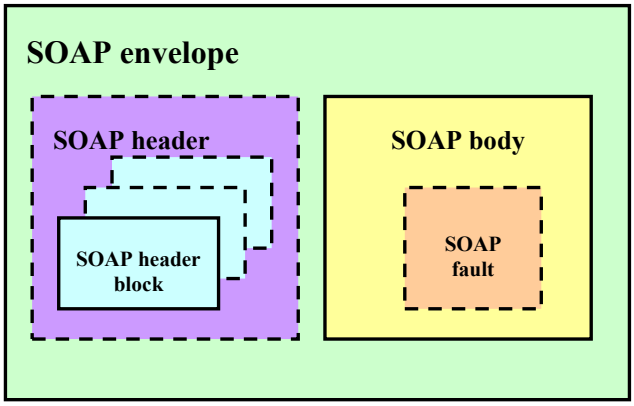


Figure 3.9: SOAP message

A SOAP message consists of a *SOAP envelope*, which encloses an optional *SOAP header* and a required *SOAP body* as can be seen in Figure 3.9. The *SOAP body* holds the actual data *(message payload)* to be transferred to an ultimate receiver and hence is a mandatory *element* of a SOAP message. It may have any number of children constituting the payload. In case of errors, the *SOAP body* contains a single *SOAP fault* as a means for carrying error information within a SOAP message. The *SOAP header*, on the other hand, is a means of extension. It consists of zero or more *SOAP header block*s, which contain user-defined extensions such as authentication, digital signatures, transaction support, priority information, etc. The *SOAP header* may be intended for any receiver along the path of the SOAP message.

An example of a SOAP message is given in Figure 3.9. The message contains a *SOAP header block* with a local name of `subscriptioncheck` and a body element with a local name of `subscriber`. The *header* in this example contains priority and expiration information that may be used by an intermediary to prioritize the delivery of the message. Actual payload of subscriber information is found in the *SOAP body*.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:subscriptioncheck xmlns:n="http://example.org/subscriptioncheck">
   <n:priority>1</n:priority>
   <n:expires>2004-01-31</n:expires>
  </n:subscriptioncheck>
 </env:Header>
 <env:Body>
  <s:subscriber xmlns:s="http://example.org/subscriber">
   <s:name>Ayisigi Basak Sevdik</s:name>
   <s:subscriberID>012345</s:subscriberID>
  </s:subscriber>
 </env:Body>
</env:Envelope>
```

Figure 3.10: SOAP message containing a SOAP header block and a SOAP body

There are also *SOAP attributes*, which may be optionally found in the *SOAP envelope*, the *SOAP body*, *SOAP header*, as well as *SOAP header blocks*. These *attribute*s can be one of *SOAP encodingStyle*, *SOAP role*, *SOAP mustUnderstand*, and *SOAP relay attribute*s. The first one, indicates the encoding rules used to serialize parts of a SOAP message, and it may appear only on a *SOAP header block*, or a child *element*

of a *SOAP body* which is not a *SOAP fault*, or a child *element* of a *SOAP detail* which is a child of the *SOAP fault*. The *SOAP role attribute* indicates the SOAP node to which a particular *SOAP header block* is targeted, and the *SOAP mustUnderstand attribute* shows whether the processing of a *SOAP header block* is required or optional. Finally, the *SOAP relay attribute* indicates whether a *SOAP header block* targeted at a *SOAP receiver* is to be relayed if it's not processed.

3.2.1. SOAP 1.2 Part 1

SOAP 1.2 Part 1 defines the SOAP messaging framework, consisting of the SOAP processing model, the SOAP extensibility model, the SOAP protocol binding framework, and the SOAP message construct. The SOAP processing model defines the rules for processing a SOAP message, whereas the SOAP extensibility model defines the concepts of *SOAP feature*s and *SOAP model*s. The third item of Part 1, i.e. the SOAP protocol binding framework, gives the rules for defining a binding to an underlying protocol (which can - but not necessarily - be HTTP) to be used for exchanging SOAP messages. The SOAP message construct, on the other hand, describes the structure of a SOAP message. Now, let's look at these constituents in more detail.

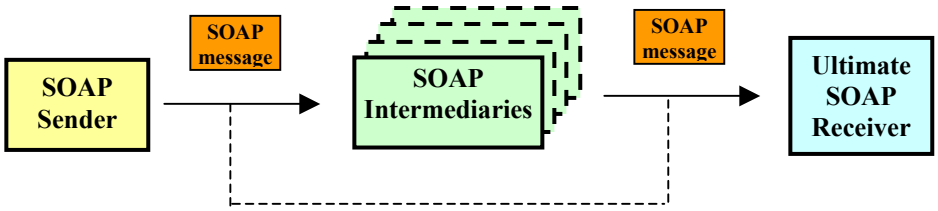


Figure3.11: SOAP Processing Model

3.2.1.1 SOAP Processing Model

The SOAP processing model provides a distributed model where a SOAP message is sent by a "*SOAP sender*" (client-side) to an "*ultimate SOAP receiver*" (server-side) with or without going through one or more "*SOAP intermediaries*" as seen in Figure 3.11.

The model mainly specifies how a *SOAP receiver* (server-side) processes a SOAP message.

The process model describes processing of only a single SOAP message. If subsequent or multiple processing is used by a *SOAP feature*, then it is the responsibility of this *feature* to process the messages accordingly (in a combined manner).

Each *SOAP sender*, *ultimate SOAP receiver*, and *SOAP intermediary* is a SOAP node. These SOAP nodes act in one of the predefined *SOAP role*s, which are identified by URIs (Universal Resource Identifiers), also called *SOAP role name*s. There are three basic *SOAP role*s defined in the SOAP 1.2 specification: *next*, *none*, and *ultimateReceiver*. The first one is the role that each *SOAP intermediary* and *ultimate SOAP receiver* must perform. The *ultimateReceiver*, as can be understood from its name, is the role for the *ultimate receiver*, and *none* is a role which none of the SOAP nodes should act in, as a *SOAP header block* with a role of *none* is not formally processed. Such *header block*s may contain information on how other *header block*s should be processed. Other *SOAP role name*s from the ones defined in the specification may also be used. It is important to note that, there is no routing or message exchange semantics related with *SOAP role name*s.

Short-name	Name	Description
`next`	`"http://www.w3.org/2003/05/soap-envelope/role/next"`	Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
`none`	`"http://www.w3.org/2003/05/soap-envelope/role/none"`	SOAP nodes MUST NOT act in this role.
`ultimateReceiver`	`"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"`	The ultimate receiver MUST act in this role.

Table 3.1: SOAP Roles defined by the SOAP 1.2 specification

The processing of a SOAP message is done by first determining the roles in which the node is to act and then identifying the mandatory *header block*s of the message targeted at the current node. If the node does not understand any of these blocks, a fault

is to be generated and further processing is to be prevented. Otherwise, all mandatory *header block*s targeted for the node must be processed. The node may also choose to process non-mandatory *header block*s targeted at it. If the current node is the *ultimate receiver*, then the *SOAP body* must also be processed. Else, if the node is an *intermediary*, then it must relay the SOAP message if the result of processing and SOAP MEP (Message Exchange Pattern) require the message to be sent further along its path. A SOAP message processing may only result in one fault and in the case of multiple errors, the node chooses one to form a fault.

3.2.1.2 SOAP Extensibility Model

SOAP is designed to be a simple protocol, however it allows the use of some extensibility mechanisms to provide additional capabilities. The SOAP extensibility model specifies these mechanisms and the way they are used.

A *SOAP feature* is an extension to the SOAP messaging framework such as reliability, security, correlation, routing, message exchange patterns (MEPs), etc. The SOAP extensibility model explains two mechanisms to express *feature*s. One of these mechanisms is the SOAP processing model, which enables *feature*s to be expressed inside the *SOAP envelope* as *SOAP header blocks*. These *header block*s, as previously mentioned, can be targeted at any SOAP node which has the mechanisms necessary to implement the specified *feature*s. The combined syntax and semantics of more than one *header block* is referred to as a '*SOAP module*'. A *SOAP module* must clearly and completely specify the content and semantics of the *SOAP header block*s used to implement the behavior in question, including if appropriate any modifications to the SOAP processing model. It must clearly specify any known interactions with or changes to the interpretation of the *SOAP body*. Furthermore, it must clearly specify any known interactions with or changes to the interpretation of other *SOAP feature*s and *SOAP module*s. For example, we can imagine a module that encrypts and removes the *SOAP body*, inserting instead a *SOAP header block* containing a checksum and an indication of the encryption mechanism used. The specification for such a *module* would indicate that

the decryption algorithm on the receiving side is to be run prior to any other *module*s, which rely on the contents of the *SOAP body*.

The other *feature* expressing mechanism is a SOAP protocol binding. A SOAP protocol binding operates between two adjacent SOAP nodes along a SOAP message path. There is no requirement that the same underlying protocol is used for all hops along a SOAP message path. In some cases, underlying protocols are equipped, either directly or through extension, with mechanisms for providing certain *feature*s. The SOAP Protocol Binding Framework provides a scheme for describing these features and how they relate to SOAP nodes through a *binding specification*. Certain *feature*s might require end-to-end as opposed to hop-by-hop processing semantics. Although the SOAP Protocol Binding Framework allows end-to-end features to be expressed outside the *SOAP envelope*, no standard mechanism is provided for the processing by intermediaries of the resulting messages. A *binding specification* that expresses such *feature*s external to the *SOAP envelope* needs to define its own processing rules for those externally expressed *feature*s. A SOAP node is expected to conform to these processing rules (for example, describing what information is passed along with the SOAP message as it leaves the *intermediary*). The processing of *SOAP envelope*s in accordance with the SOAP Processing Model must not be overridden by *binding specification*s.

A Message Exchange Pattern (MEP) is a *template* that establishes a *pattern* for the exchange of messages between SOAP nodes. MEPs are a type of *feature*. Underlying protocol *binding specification*s can declare their support for one or more named MEPs [20].

3.2.1.3 SOAP Protocol Binding Framework

SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a *binding*. The SOAP Protocol Binding Framework provides general rules for the specification of protocol

bindings; the framework also describes the relationship between *binding*s and SOAP nodes that implement those *binding*s. A SOAP binding specification:

- declares the *feature*s provided by a *binding*,
- describes how the services of the underlying protocol are used to transmit SOAP message infosets,
- describes how the services of the underlying protocol are used to honor the contract formed by the *feature*s supported by that *binding*,
- describes the handling of all potential failures that can be anticipated within the *binding*, and
- defines the requirements for building a conformant implementation of the *binding* being specified.

The specification describes the SOAP HTTP *binding* in Part2. Other underlying protocols can also be used.

3.2.1.4 Security Considerations

The SOAP Messaging Framework does not directly provide any mechanisms for dealing with access control, confidentiality, integrity and non-repudiation. Such mechanisms can be provided as *SOAP extension*s using the SOAP extensibility model.

SOAP implementors should consider SOAP applications sending intentionally malicious data to a SOAP node. The SOAP specification recommends that a SOAP node receiving a SOAP message is capable of evaluating to what level it can trust the sender of that SOAP message and its contents. Thus, only carefully specified *SOAP header block*s, as well as body contents, with well understood security implications of any side effects should be processed by a SOAP node. Special attention should be paid to the security implications of any data in a SOAP message as malicious data might trigger remote execution of some actions in the receiver side.

SOAP intermediaries present an opportunity for man-in-the-middle attacks. A compromised *SOAP intermediary* can trigger a wide range of potential attacks. Thus, security issues must be taken seriously when using SOAP.

3.3 Web Services Description Language (WSDL)

Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered. [28]

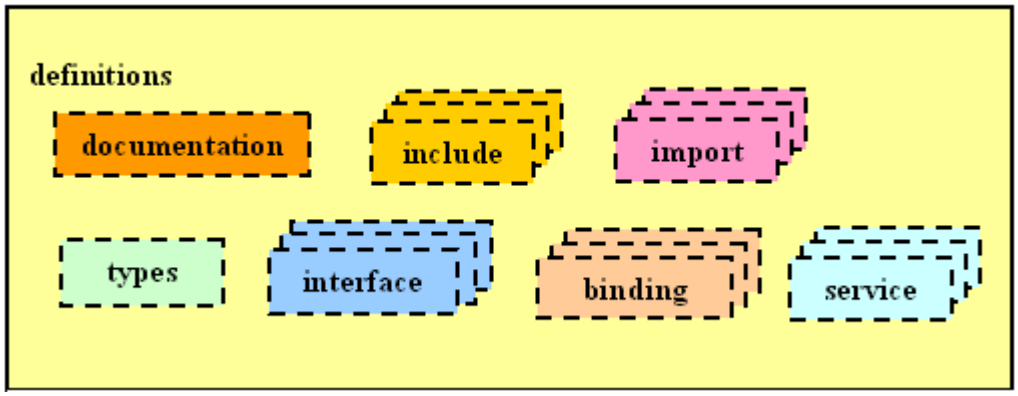


Figure 3.12: Structure of a WSDL document

WSDL 1.0 has been developed by IBM, Microsoft and Ariba. WSDL 1.1 was submitted to the W3C as a suggestion for describing services for the W3c XML Activity on XML Protocols in March 2001. The first working draft of WSDL 1.2 was released by W3C in July 2002. The latest version of the working draft is WSDL 2.0 and was released in November 2003. A WSDL document is a simple XML document which describes a Web service. It basically specifies the location and the methods of the service. A WSDL document uses the following main *element*s to establish this: *interface*, *service*, *types* and *binding*. It can make use of other *element*s, such as an *include element* and an *import element*, which enables the combination of the definitions of several Web services in one single WSDL document.

The most important WSDL *element* is the *interface* as it defines a Web service, the operations available, and the messages involved. The *service element*, on the other hand, defines the data elements of an operation. It can have one or more parts. The data types used by the Web service are defined by the *types element*, as can be understood from its name. In order to achieve platform independence, WSDL uses XML Schema syntax for writing this *element*. Finally, the *binding element* indicates the message format and protocol details for each endpoint (port). Below an example of a WSDL document is given.

The *endpoint* gives the connection point to a Web service. In the above example, the *endpoint* is identified in the *interface element* by the name `glossaryTerms`. Here, the *interface element* also gives information about the *operation*s that can be performed by the Web service and the messages involved in using this Web service. The available *operation* of the Web service in the example is `getTerm`, and the two messages involved are `getTermRequest` and `getTermResponse`. The `getTerm` *operation* in this example is of type *request-response*. There are three other *operation*s types in WSDL. These are *one-way*, *solicit-response*, and *notification*. The example also has a *binding element* with two *attribute*s of *name* and *type*. The *name attribute* specifies the name of the *binding*, which is `b1` in the example, and the `type` *attribute* specifies the port for the *binding*, which is `glossaryTerms` port in this case. The *binding element* has the *soap:binding element*, indicating that it's a *SOAP binding*, and the *operation element* as its children. The *binding* can also be HTTP or MIME *binding*, apart from *SOAP binding*. The former has *style* and *transport attribute*s. The *style attribute* of the *soap:binding* is *document* in this example, however it can also be *rpc*. The *transport attribute*, on the other hand, specifies which protocol to use, and is defined as *HTTP* in the example. The latter, i.e. the *operation element*, defines the *operation*s that the port enables. The corresponding SOAP action for each *operation* and input/output encoding method needs to be defined . The encoding methods in the example are given as *literal*.

```
<message name="getTermRequest">
      <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
      <part name="value" type="xs:string"/>
</message>
<interface name="glossaryTerms">
   <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
   </operation>
</interface>
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation>
         <soap:operation
           soapAction="http://example.com/getTerm"/>
         <input>
      <soap:body use="literal"/>
         </input>
         <output>
      <soap:body use="literal"/>
         </output>
   </operation>
</binding>
```

Figure 3.13:An example of a WSDL document

WSDL describes a Web service in two fundamental stages: one abstract and one concrete. Within each stage, the description uses a number of constructs to promote reusability of the description and separate independent design concerns.

At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are described independent of a specific wire format using a type system, typically *XML Schema* [28].

An *operation* associates a message exchange pattern with one or more messages. A *message exchange pattern* identifies the sequence and cardinality of messages sent and/or received as well as who they are logically sent to and/or received from. An

interface groups together *operation*s without any commitment to transport or wire format.

At a concrete level, a *binding* specifies transport and wire format details for one or more interfaces. An *endpoint* associates a network address with a *binding*. And finally, a *service* groups together *endpoint*s that implement a common interface.

3.4 Universal Description, Discovery and Integration (UDDI)

In the previous section we talked about a language to describe Web services, but nothing has been said about the businesses providing these services and how to get in touch with these businesses to make use of their services. This is where a need for Web-based service registries and hence for Universal Description, Discovery and Integration (UDDI) arises. Universal Description, Discovery and Integration, or UDDI, is the name of a group of web-based registries that expose information about a business or other entity and its technical interfaces (or API's) [23]. These registries constitute a means of providing interoperability between buyers, suppliers, marketplaces, service providers and other entities. Anyone or any business can freely publish information about themselves and their services, search for a specific service they need, find out who is providing a desired service and how they can interact with that service provider through the use of UDDI registries.

UDDI has been developed initially by the joint efforts and assembly of 36 companies to form its sponsoring organization, UDDI.org, and release the first version (1.0) of its specification in September 2000. Today more than 200 companies are supporting the initiative including IBM, Microsoft, Ariba, Oracle, SAP, Sun, Commerce One, Boeing and Ford. The latest version is UDDI version 3.0.1 released November 2003 [24], for which UDDI.org has started working jointly with the open standards organization OASIS.

UDDI supports three kinds of information that are called *white pages*, *yellow pages*, and *green pages*, analogous to a phone directory. *White pages* contain basic information about the business, such as name, contact details and sometimes identifiers like tax IDs to uniquely identify the business. *Yellow pages* include categorized information about the services provided by a business. Here, a service can be categorized under multiple groups. Finally, *green pages* present technical information about a service provided by a business including service location, various interfaces, specifications and other data needed to run the Web service.

Since it's just another Web service related specification, UDDI is based on XML. Thus, a UDDI registry is made up of XML-based service descriptors. Figure 3.15 displays the four main data structures, or XML *element*s, composing a UDDI Registry as described in UDDI Version 3.0.1. They are the *businessEntity*, the *businessService*, the *bindingTemplate* and the *tModel*.

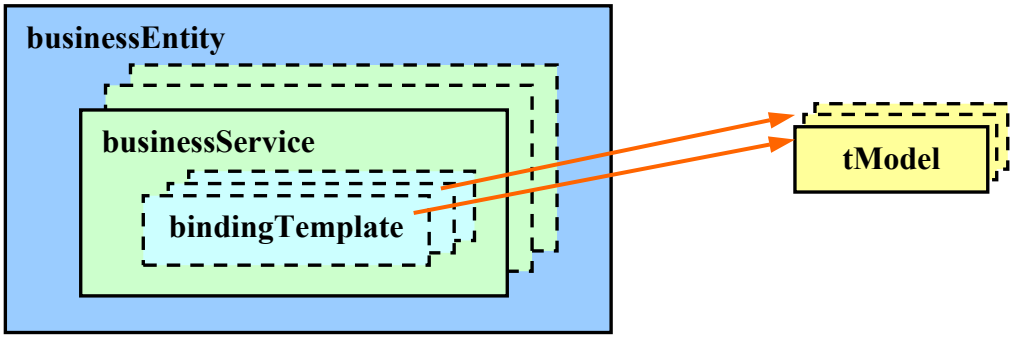


Figure 3.14: UDDI Registry Main Data Structures

A *businessEntity* provides information about a business or organization within the UDDI. Information about provided services and technical information is represented by *businessService* and *bindingTemplate* elements within a *businessEntity*. Each *businessService* gives descriptive information including the purpose of a Web service offered by the business or organization. It may contain *bindingTemplate*s, which present technical information about the Web service described. A *tModel*, on the other hand, provides descriptions of specifications, transports, protocols or namespaces pertaining to a Web service, in order to enable compatibility checking and interaction guidance with

this Web service in question. An important point to be made here is that *tModel*s are separated from the *businessEntity-businessService-bindingTemplate* containment relationship, hence they are only referenced by *bindingTemplate* elements. This enables the reuse of *tModel*s by different *bindingTemplate*s, and such *bindingTemplate*s that refer to the same *tModel* or set of *tModel*s are said to have the same "technical fingerprint" or be of the same type.

The *tModel* mechanism serves a useful purpose in discovering information about interfaces and other technical foundation concepts that are exposed for broad use by an individual service or registration instance [23]. An exemplary use would be to find compatible business partners for your business, which has the ability to accept electronic orders, by consulting one of the public UDDI sites. This can be achieved by looking up the service in discussion, and locating those businesses that have previously advertised support for doing electronic commerce that is compatible with your abilities. When you register to a UDDI, a *tModel* and a corresponding *tModelKey* providing the interface or specification for your e-commerce capability also gets registered. The compatible partner capabilities are kept within the UDDI through service *binding*s, which reference this *tModel*. As the example underlines, *tModels* provide a common reference point that allows software companies to register a technical interface and identify compatible implementations. The advantage that *tModel*s provide for businesses that seek services is less work in finding the compatibility of a service advertised by a business partner with its own system.

Now that the basic structure of UDDI along with its uses has been presented, let us look at a UDDI listing (Figure 3.16). This listing was taken from the Microsoft UDDI Business Registry (UBR) Node. It displays information about the online order placement service of the company VB Web services Inc. As the *businessEntity* description states, this is a fictitious company for testing UDDI. A single *businessService* of the name `online order placement` is provided, which enables placing an order with the company online. The *bindingTemplate* specifies an access point and explains that the orders are placed via SOAP over HTTP.

```
<?xml version="1.0" encoding="utf-8" ?>
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   businessKey="144f7530-2c61-410d-aa6c-cfd4d840ba8b" operator="Microsoft
   Corporation" authorizedName="Yasser" xmlns="urn:uddi-org:api_v2">
     <discoveryURLs>
         <discoveryURLuseType="businessEntity">http://uddi.microsoft.com/
            discovery?businesskey=144f7530-2c61-410d-aa6c-cfd4d840ba8b
            </discoveryURL>
     </discoveryURLs>
     <name xml:lang="en">VB Web services Inc.</name>
      <description xml:lang="en">Fictitious Company For Testing  UDDI
      </description>
     <businessServices>
        <businessService serviceKey="42f82803-c697-4ca7-aca1-
         6efa6905b7de" businessKey="144f7530-2c61-410d- aa6c-
         cfd4d840ba8b">
          <name xml:lang="en">Online Order Placement</name>
          <description xml:lang="en">Place an order with our company
            online</description>
          <bindingTemplates>
              <bindingTemplate bindingKey="ad63f1a6-1d2c-4656-baf3-
                 8e889a80126b" serviceKey="42f82803-c697-4ca7- aca1-
                 6efa6905b7de">
                  <description xml:lang="en">Place orders via SOAP over
                     HTTP</description>
                   <accessPoint URLType="http">http://www.vbws.com/
                     ordering.asp </accessPoint>
                  <tModelInstanceDetails>
                      <tModelInstanceInfo tModelKey="uuid:4cd7e4bc-648b-
                        426d-9936-443eaac8ae23" />
                      </tModelInstanceDetails>
              </bindingTemplate>
          </bindingTemplates>
      </businessService>
     </businessServices>
</businessEntity>
```

Figure 3.15: An Example UDDI Listing

Currently four companies are operating a public instance UDDI registry, also called Universal Business Registry (UBR). Each of these companies (IBM, Microsoft, SAP, and NTT Com) is operating a different UBR node. Table 3.2 displays the URLs for these UBR nodes.

UBR Node Name	Homepage	Inquiry API	Publish API
IBM UBR Node	http://uddi.ibm.com/	http://uddi.ibm.com/beta/inquiryapi	https://uddi.ibm.com/beta/publishapi
Microsoft UBR Node	http://uddi.microsoft.com/	http://uddi.microsoft.com/inquire	https://uddi.microsoft.com/publish
SAP UBR Node	http://uddi.sap.com/	http://uddi.sap.com/uddi/api/inquiry	https://uddi.sap.com/uddi/api/publish
NTT Com UBR Node	http://www.ntt.com/uddi/	http://www.uddi.ne.jp/ubr/inquiryapi	https://www.uddi.ne.jp/ubr/publishapi

Table 3.2: UDDI Business Registry (UBR) Node URLs

These registries replicate with each other to provide access to the whole UBR. Here, one can look up Web services interfaces and technical details, find out potential partners, or advertise services by registering for free. There are also browsing capabilities available through taxonomic classification of UDDI entities. This enables the discovery of a desired service, or business without having to know specific information about it, such as its full name or *business key*. Several categorization schemes, such as United Nations Standard Products and Services Code System (UNSPSC), North American Industry Classification System and ISO 3166 Geographic Classification System, are available.

There may be some security concerns related to the use of UDDI registries. Companies may want to restrict access to their Web services to only trusted parties. Although missing in earlier versions of the UDDI specification, these security concerns are considered in UDDI Version 3.0.1, where some security policies are defined for nodes, registries and clients. Several optional and extensible mechanisms for this purpose are described. For example, one or more identification systems may be used in an implementation of a UDDI node for authorization or data access restriction purposes. Also UDDI Version 3.0.1 Specification states that XML Digital Signatures are supported on UDDI data so that a client or business can verify the integrity of the data with respect to the publisher.

Today, private registries are being actively used, whereas use of public registries is more limited. [25] However, as UDDI evolves with new versions, as the missing pieces are put in, like security policies to establish majority's trust, and as customers become aware of these developments, it seems UDDI is sure to achieve greater acceptance and use.

Chapter 4
XML in Electronic Government

Electronic government is defined as "the uninterrupted and secure, mutual execution of the duties and services the government is responsible to perform towards citizens, and the duties and services of citizens towards the government, in electronic communication and processing environments" [2]. The goals of transferring the traditional government into such an electronic government can be listed as follows:

- transparency of the government,
- fast and efficient execution of the government,
- increased citizen participation in governance at every level,
- prevention of work and data recursion by inter-organizational data exchange,
- making life easy for citizens that the government serves, and
- improved and fast decision-making process based on information by authorities.

Over the past few years XML has gained broad acceptance in the business community especially for providing Web services, thus making it a possible technology

to be employed by the electronic government to achieve its aims. In this chapter, we are exploring the use of XML in electronic government.

How can XML be applied to achieve such an electronic government? What would be the benefits of adopting XML for e-government use? In the following subsections, a deeper look will be taken in order to answer these questions.

4.1 The Benefits of e-Government XML Adoption

The Extensible Markup Language (XML) is a flexible, nonproprietary set of standards for tagging information so that it can be transmitted over a network such as the Internet and readily interpreted by disparate computer systems [27]. XML can make it easier to discover, obtain and process information dispersed among different organizations or systems of an e-government. This is a key feature an electronic government requires, since a tremendous amount of information flow takes place between the different government departments and agencies to accomplish inter-governmental tasks. Inter-governmental tasks consist of a government agency to request information from other agencies, process this information according to its own needs, or discover and send information to another agency that needs it. By tagging data, XML enables these disparate systems of different government agencies to easily interpret the data produced by other departments. Moreover, this necessity to exchange data is not limited other inter-governmental tasks, nor is it only between different government agencies. Government organizations exchange data with other external entities including private organizations and other governments. XML enables interfacing these disparate systems and locating and sharing data among them. Thus, it makes data transmission to these external entities possible and allows the data to be readily interpreted by these different systems due to its property of tagging data descriptively and making it machine-readable. Furthermore, the government can conduct transactions based on this data exchange, which is a key element for the provision of government-to-citizen and government-to-business services. All these promote the government a step further in

achieving its goal of fast and efficient execution of its services, as well as, preventing inter-organizational data recursion.

One other feature of XML is that it is platform-independent. It can operate on any combination of computer hardware and XML-enabled software [27]. Thus, by using XML in electronic government, the need to provide customization and new infrastructure to build e-government services is removed. XML supports Internet-based data exchange, so the necessary communications infrastructure needed for e-government is already in place. This frees the government of building new interoperable systems and data communications infrastructure all over its architecture, which could add up to be a significant amount in costs. Hence by adopting XML rather than another technology, an electronic government can significantly lower its initial startup costs.

On top of these, XML is a license-free open standard supported by a wide variety of organizations. It is already accepted by the business community. According to a survey published by the Giga Information Group [26] (as reported by the U.S. General Accounting Office [27]) about eighty percent of businesses have begun utilizing XML in their organizations. Also, a variety of software supporting XML and its applications has been and is being developed. This makes XML more suitable for widespread adoption, thus makes the broad implementation of consistent tagging of data more of a possibility.

XML technology is reusable. Once it is written, both its creators and external parties can potentially reuse XML code. This prevents redundant overlapping code writing of different governmental agencies and recursion of work. Agencies can easily reuse XML written by another agency or organization, to perform common tasks. An example of a such a common task can be the task of handling financial transactions. Most government departments need to handle such transactions when charging citizens or businesses for a specific service, or when buying necessary services or products from external entities for governmental use. The basics of a financial transaction are the same and independent from the purpose of use. Thus, the same XML code can be reused at every government agency that requires handling a financial transaction for its services.

Another important property of XML is its extensibility. This feature facilitates interaction among a variety of devices. It allows the same XML document to be displayed on a variety of devices from personal computers to palmtops, by interpreting the document through different stylesheets. Such a property can be made great use of in the interactions of the government with citizens, business, or even other government agencies.

The extensibility property of XML, also has triggered the development of data vocabularies (or languages) designed to meet the needs of specific businesses an professions. Examples include *electronic business XML (ebXML)*, *Legal XML*, *Human Resources XML (HR-XML)*. These could be utilized by the government in providing functions that are specific to certain disciplines. For example Legal XML can be made use of by the justice department.

4.2 Pitfalls of e-Government's XML Adoption

All of the benefits of adopting XML use presented above, enables improved and fast decision-making process based on information by the government as they help create a more productive, efficient execution of government services. Thus, the overall improvement of government services and the ability to access them online through a variety of devices would greatly simplify government-related processes for both citizens and businesses, as well as increase their participation.

Although making use of XML in the electronic government arena provides many benefits and supports the original goals of e-government, XML has its disadvantages. There are some additional precautions to be taken and considerations to be made in order to utilize XML in its fullest potential. Some of the pitfalls associated with implementing XML that first come to mind can be listed as: [27]

- the risk that redundant data definition, vocabularies, and structures will contradict (proliferate),
- the potential for proprietary extensions to be built that would defeat XML's goal of broad interoperability, and
- the need to maintain adequate security.

The first pitfall is the risk of creating contradicting data definitions, vocabularies or structures. If efforts are not guided, several data definitions or vocabularies can be created for the same purpose, thus, most of these definitions would be redundant. Furthermore, the unsupervised creation of such definitions can cause incompatibilities.

A second downside arises from the fact that the extensibility property of XML can be misused. Software vendors and system developers may be tempted to add proprietary extensions to the XML standards when they build specific systems [27], limiting the systems abilities to freely exchange information with other systems supporting XML.

Another pitfall might be the fact that security standards of XML are not as mature as they should be. The need for intense security measures at certain parts of the governmental work cycle is unquestionable. Hence, the immaturity of XML's security standards would be problematic. However, there is intense on-going work in standards development, and the shortcomings in the standards are being overcome through these improvements. Adoption of these newly developed security measures by XML vendors should make security become less of a concern in the near future.

Looking further at the downsides of utilizing XML, another disadvantage to be considered might be the burden of having to build XML namespaces (vocabularies) and repositories for separate areas of interest. However, though costly at first, it seems a fair price to pay when you are achieving features like interoperability as well as marking up your data with interest-specific description providing semantics in the electronic government.

An important point to be reviewed in the electronic government's adoption of XML is that the government should adopt business standards that best achieve widespread acceptance. There are a variety of XML standards currently being developed by different standard-setting organizations in the private sector. However, there is no standard that has achieved such a widespread acceptance and it is not clear which standards will prevail in the long run. Thus, if the government bases its applications on any of the current proposals, these applications may be incompatible with future standards.

It has been said that e-government can make use of industry-specific XML vocabularies. However, such vocabularies tailored to address specific industries and business activities are still mostly in the initial stages of their development. Thus, the government-wide adoption of these vocabularies is not currently possible. Again, there is on-going development and although they are not yet ready for government use, they will be in the near future.

4.3 Points to be Considered in e-Government's XML Adoption

Taking the benefits and the pitfalls of electronic government's XML adoption into account, a conclusion stating that "the e-government has to make some considerations when adopting XML to prevent itself from being affected by the disadvantages of using XML" would not be wrong.

One of these considerations should be the requirement of establishing a government-wide strategy for XML adoption. If the government does not define a strategy in utilizing XML, government agencies may built their own XML-based systems taking advantage of XML's flexibility, but these systems might not be compatible with each other. Such a situation would defeat the goal of widespread data access and exchange. Thus, the government, after establishing a centralized registry of key XML data elements and structures, should direct the use of these elements and structures in XML system development taking place in its different agencies. In this way, data structures could be reused lowering costs and system interoperability could be guaranteed.

As previously stated, it is important for the government to build upon already developed business standards and vocabularies. However, it is important that the government utilizes such standards after they have completed their development and have attained wide acceptance.

Finally, referring back to the idea of defining a government-wide strategy, in order to make maximal use of XML's benefits, government agencies must implement XML through enterprise architectures. Effective XML implementation depends on complete and well-established data definitions and structures, which can be best obtained through the process of defining and adopting an enterprise architecture [27]. Otherwise, if an overall strategy is not accepted, the need for reworking agency systems might arise in the future, causing additional costs.

To summarize, the electronic government can profit immensely from XML adoption if it takes a few precautions first to forestall implications of XML's downfalls. The goals of making the government more productive and efficient, increasing citizen participation, improved decision-making process, preventing redundant recursion of work can all be established by choosing XML as a technology for electronic government. Furthermore, careful planning and some precautions in certain delicate areas can easily prevent pitfalls.

Chapter 5
Exemplary Scenarios

In the previous chapter, we have discussed how XML can be used in electronic government. In this chapter, we would like to present some scenarios which are likely to take place in an electronic government. Here, it is intended to present possible scenarios covering all three e-government aspects of government-to-citizen services, government-to-government services, and government-to-business services.

5.1 Scenario 1

Our first exemplary scenario defines a Citizen A applying for a driver's license. This application may either be a first time application or a renewal application. In such a case, the first step Citizen A takes is to fill out the application form. This is an electronic form to which Citizen A is directed to by the government portal when s/he makes the appropriate choices and follows the appropriate links towards his/her wish to apply for a driver's license. The form request the minimal amount of information to perform the desired service from Citizen A. Citizen A enters his/her name, social security number, tax number, and the name of the hospital that has his/her health records.

After Citizen A submits the requested information through the electronic form, the government begins executing the citizen's service request. The next few steps are exemplary of government-to-government services. Here, the government performs certain inquiries within its departments to make sure that Citizen A is eligible to get a driver's license. As it is a service of the Traffic Division of the General Directorate of Security to provide the requested service of providing driver's licenses, the request is directed to this division.

The Traffic Division, first checks Citizen A's criminal record from the Department of Justice's database. Due to the fact that this scenario takes place in an electronic government, Department of Justice provides the functionality to search criminal records in its database. If the criminal record of Citizen A does not pose an obstacle against getting a driver's license, then the Traffic Division goes on to check his/her health record. It is assumed here that hospitals are all connected to the Ministry of Health, and health records of a certain patient at certain hospital database can be obtained by authorized government agencies through a service provided by the Ministry of Health. If the obtained health records present no disabling properties, the Traffic Division continues its investigation. If the license is being renewed, then the division checks for the citizen's traffic violations within its own database.

If all of the investigated properties comply with the requirements of obtaining a driver's license, then the Traffic Division assigns an exam date for Citizen A. This date may either be immediately reported to the citizen while s/he is online, after s/he submits the form and the necessary processing is done, or it may be reported via e-mail, fax, etc. After Citizen A takes the exam and successfully passes, s/he is charged for the service provided using the tax payer's information provided by the Tax Office. It should be noted that, in case of a renewal, Citizen A may not have to take the exam. In such a case, the exam date assignment step is skipped and the Traffic Division directly charges the citizen for the service. Then, the Traffic Division marks Citizen A's driver's license as valid, and finally assigns and reports a date for Citizen A to pick up the new license.

5.2 Scenario 2

The second scenario consists of two parts:

a. Citizen A commits a traffic violation.

b. Citizen A pays his/her traffic violation fine.

5.2.1 Part A

In the first part of scenario 2, Citizen A commits a traffic violation. Let's say s/he gets a ticket for speeding. The officer on duty, who gave the ticket to Citizen A, reports this to the Traffic Division and the division immediately updates Citizen A's information to add the ticket. Then, the Traffic Division reports Citizen A's insurance company of the ticket. It should be noted that an ideal electronic government environment is envisioned here. Thus, as part of its government-to-business services, the Traffic Division informs the insurance company through e-mail, fax, etc. Finally, the insurance company takes the appropriate action upon receiving this information. This may be an increase in Citizen A's insurance payments or removal of his/her perfect driving discount, etc.

5.2.2 Part B

The second part of this scenario involves Citizen A paying his/her traffic violation fine using the government portal. The government portal directs Citizen A to the right destination to fulfill his/her request to pay the fine. Here, Citizen A may view his/her violations record and choose the violation that s/he wishes to pay the fine for. Then, the system asks for his/her credit card information. Citizen A enters the requested information and the Traffic Division handles the transaction with Citizen A's bank to draw the necessary amount from Citizen A's credit card. Here, it is also a possibility for Citizen A to transfer the fine fee from his/her bank. In such a case, the bank would handle the transaction with the Traffic Division. After the completion of the transaction,

the Traffic Division marks Citizen A's traffic fine as "PAID" and reports the successful completion of the service to Citizen A, if s/he is receiving the service online.

In this scenario, the online interaction of Citizen A and the Traffic Division compose an example of government-to-citizen services. The interaction between the bank and the Traffic Division, on the other hand, constitute an example of government-to-business services.

5.3 Scenario 3

The third scenario involves the car tax payment of Citizen A. Citizen A wishes to pay his/her annual car tax fee. Again, from the government portal Citizen A is directed to the appropriate destination providing the desired service. Here, Citizen A enters his/her credit card information. The Tax Office requests the withdrawal of the necessary amount for Citizen A's annual car tax from Citizen A's bank account for the specified credit card via a transaction. As soon as the Tax Office and the bank complete the transaction, Citizen A's car tax is marked as "PAID" in the Tax Office database. As a final step, the Tax Office reports that Citizen A has paid his/her annual car tax fee, to the Traffic Division. Hence, the Traffic Division makes the necessary updates in its database for Citizen A's car information.

5.4 Scenario 4

In the last scenario, Citizen A desires to renew his/her car's insurance. Thus, Citizen A pays the renewal fee to the insurance company. The insurance company may provide such a service online. If this is the situation, then as soon as the insurance company completes the transaction with Citizen A's bank, it reports to the Traffic Division. Otherwise, the insurance company gets the payment directly form Citizen A, and then reports to the Traffic Division that this particular citizen has paid his/her insurance. Upon receiving this information, the Traffic Division updates Citizen A's car information in its database.

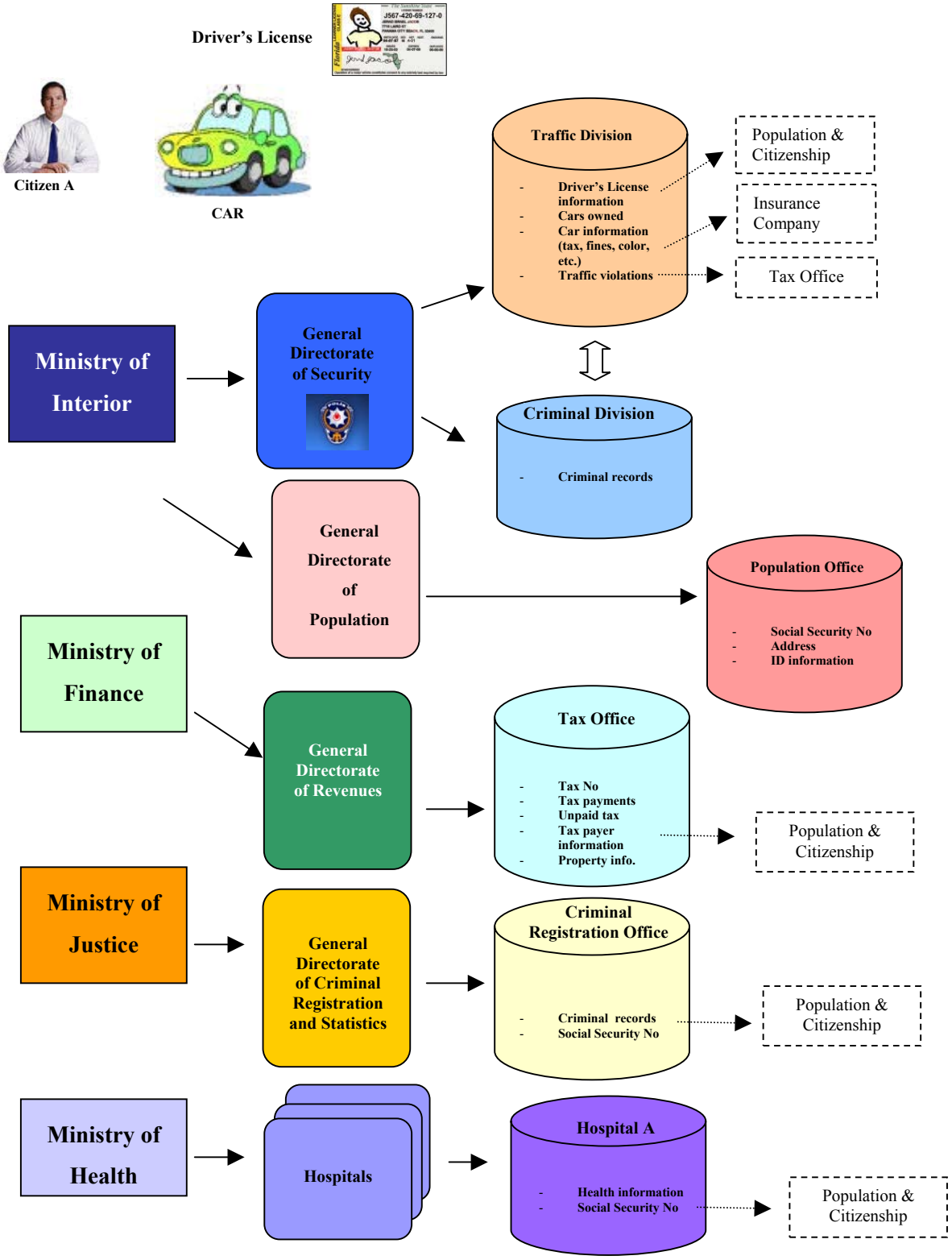


Figure 5.1: Graphical representation of related actors and databases

5.5 Related Actors and Databases

Now that we have gone over what the different scenarios involve, let us look at the related actor and databases that take part in these scenarios to establish a better understanding of the scenarios. Figure 5.1 shows a graphical representation of the related actors and databases.

5.6 Implementation of the Scenarios

There are six databases involved in these scenarios. These databases are the Traffic Division database, Criminal Division database, Population Office database, Tax Office database, Criminal Registration database, and the Hospital Registry database. All of these databases provide services for searching and retrieving records. XML and its related standards can be used to implement these interactions. In the following subsections, the use of XML standards to implement the given scenarios is demonstrated.

5.6.1 Providing Structure

All of the databases must have specific *XML schemas* associated with them. It should be reminded that, schemas are the means of establishing a standard structure for XML documents. For this purpose the *XML schemas* `TrafficDept`, `PopOffice`, `CriminalDivision`, `CriminalRegOffice`, `TaxOffice`, and `HospitalRegistry` have been created. The `TrafficDept` *schema* will be presented in detail as a sample to provide clarity of the structure described in these schemas.

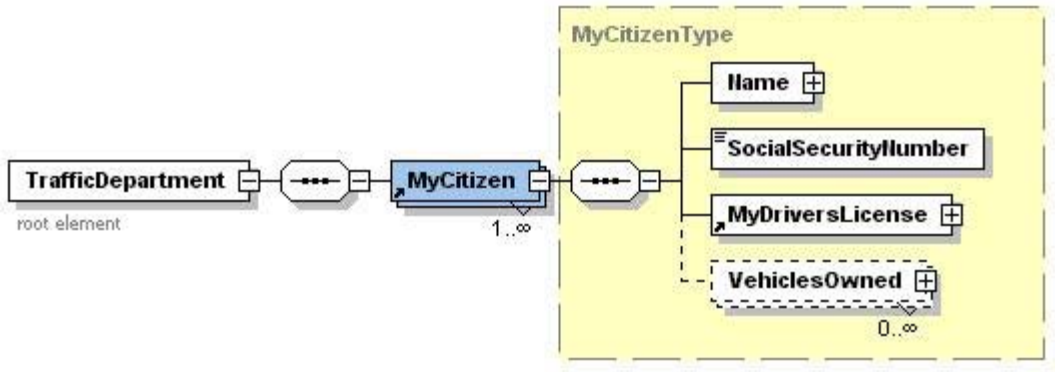


Figure 5.2: Structure of TrafficDepartment Element

The `TrafficDept` *schema* defines the structure of the Traffic Division database and records. The structure of the records in this *schema* have been defined by simulating a real-life traffic division database and the information that would be kept in its records. This structure is presented in Figure 5.2. This *schema* consists of `MyCitizen` *element*s, which simulate actual citizen records, under a *root element* with the name `TrafficDepartment`. `MyCitizen` contains the *element*s `Name`, and `SocialSecurityNumber`, which help identify the citizen. The other children of `MyCitizen` give information about the driver's license of the citizen and the vehicles s/he owns. Thus, these *element*s are named `MyDriversLicense` and `VehiclesOwned`. Considering the fact that a citizen can have ownership of more than one vehicle, the `VehiclesOwned` *element* is unbounded. It is also an optional *element*, as a citizen having a driver's license might not own a car at all.

Taking a deeper look into `MyDriversLicense` *element*, it can be observed from Figure 5.3 that it has the `LicenseNo`, `LicenseClass`, `IssuedAt`, `IssueDate`, `AdditionalEquipment_Prothesis`, and `BloodType` *element*s as its children. `LicenseNo` displays the citizen's driver's license registration number. `LicenseClass` specifies the type of driver's license owned. This can be either one of class A, B, C, D, or E as in the actual classification in licenses. `IssuedAt` and `IssueDate` *element*s

specify the county/city the license was issued in and the date it was issued at, respectively. `AdditionalEquipment_Prothesis` *element* displays information of the additional aiding equipment the driver uses, such as glasses or lenses. Finally, the last child, `BloodType` specifies the driver's blood type as group and RH type; that is, like A RH (+).

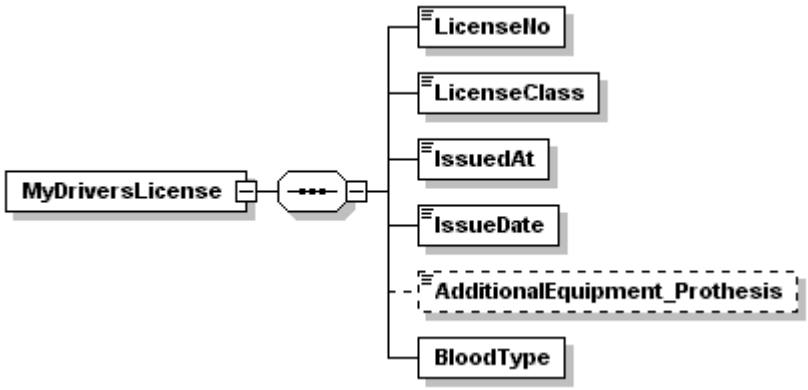


Figure 5.3: Structure of MyDriversLicense Element

The `VehiclesOwned` *element* is also composed of several child *element*s. These *element*s provide information about the specific vehicle, including license plate number (`LicensePlate` *element*), type of vehicle (`Type`), which can be one of automobile, truck, or van, usage purpose of the vehicle (`Use`) – one of private, commercial, government, or municipal, vehicle's technical properties (`TechnicalProperties`), taxes pertaining to the vehicle (`Tax`), and fines associated with the vehicle (`Fine`). Technical properties consist of the properties make, model, color, chassis serial number, motor serial number, motor type, motor horse power, motor cylindrical power, and net weight of the vehicle.

5.6.2 Sample Database Record

Now that an understanding of the basic structure of the `TrafficDept` database has been established, an example XML document representing the Traffic Division

database can be provided. Figure 5.4 displays part of such an XML document containing a citizen's traffic department record.

```
<?xml version="1.0" encoding="UTF-8"?>
<TrafficDepartment xmlns="http://my-traffic.com/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://my-traffic.com/namespace
  TrafficDept.xsd">
   <MyCitizen>
      <Name>
         <FirstName>Ayisigi</FirstName>
         <MiddleName>Basak</MiddleName>
         <LastName>Sevdik</LastName>
      </Name>
      <SocialSecurityNumber>1234567890</SocialSecurityNumber>
      <MyDriversLicense>
         <LicenseNo>961501</LicenseNo><LicenseClass>B</LicenseClass>
         <IssuedAt>Ankara</IssuedAt>
         <IssueDate>1999-05-20</IssueDate>

   <AdditionalEquipment_Prothesis>Lenses/Glasses</AdditionalEquipment_P
      rothesis>
         <BloodType Group="B" RH="+"/>
      </MyDriversLicense>
      <VehiclesOwned>
         <Type Automobile="true"/>
         <LicensePlate>06PD557</LicensePlate>
         <Use Private="true"/>
         <TechnicalProperties>
            <Make>Hyundai</Make>
            <Model>1994</Model>
            <Color>Metalic Silver</Color>
            <ChasisSerialNo>ABC12EF3456G7H8I9</ChasisSerialNo>
            <MotorSerialNo>9A87BCD6E5F</MotorSerialNo>
            <MotorType Fueled="true" Diesel="false" Euro93="false"/>
            <NetWeight>980kg</NetWeight>
         </TechnicalProperties>
         <Fine Paid="false" Name="ParkingViolation"/>
         <Tax Paid="true" Name="MotorVehiclesTaxFirstPayment"/>
         <Tax Paid="true" Name="MotorVehiclesTaxSecondPayment"/>
         <Tax Paid="true" Name="MotorVehiclesTaxThirdPayment"/>
         <Tax Paid="false" Name="MotorVehiclesTaxFourthPayment"/>
      </VehiclesOwned>
   </MyCitizen>
   <MyCitizen>
       …
    </MyCitizen>
</TrafficDepartment>
```

Figure 5.4: Sample XML document displaying a Traffic Department record

5.6.3 Providing Services

There is a need for departments or divisions to provide each other with certain services in order to successfully pursue inter-governmental interactions. In this section, an example of such a service that the Traffic Division request from the Population Office will be presented. The Traffic Division requests a citizen's record from the Population Office providing the social security number of the citizen. The Traffic Division requires this information in order to obtain citizen's address. This may be required for contact purposes. It should be noted that the Population Office provides the same service to other divisions as it is required throughout the scenarios.

The Population Office can provide this service through any means. However, it advertises this service using a WSDL document, namely `PopOffice.wsdl`. The structure of this WSDL document is presented in Figure 5.5.

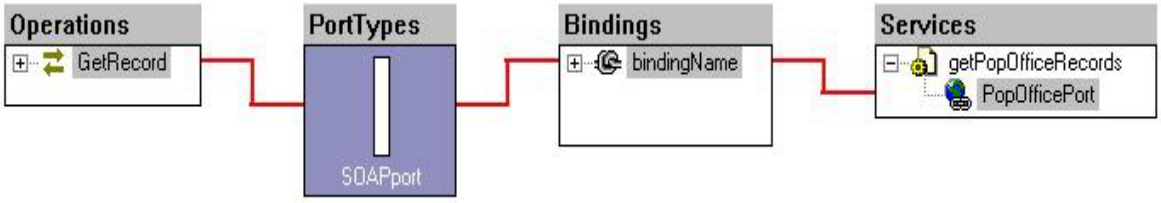


Figure 5.5: Population Office WSDL document

This WSDL document specifies that the Population Office provides a service named `getPopOfficeRecords`, which is accessible through a SOAP port named `PopOfficePort`. This SOAP port enables the `GetRecord` operation. The important properties of the port are the *location* and the *binding*. The location of the `PopOfficePort` specifies the URI that provides the named service. In this case it is given as `http://pc501b.cs.bilkent.edu.tr/cgi-bin/popSOAP3.pl`. The *binding* specifies the *port type*, the *transport URI*, and the *binding style*. The *port type* is

SOAP, and the *transport URI* is `http://schemas.xmlsoap.org/soaphttp`, which describes that the transport is to be made over HTTP. The *binding style* is defined as *rpc*, denoting remote procedure call. There are two messages related to this service; the `GetRecordResponse` and the `GetRecordRequest` messages. The request message (`GetRecordRequest`) takes a parameter of type string representing the social security number to be sent along.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <m:GetRecord xmlns:m="http://new.webservice.namespace">
         <SSN>"1234567890"</SSN>
      </m:GetRecord>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.6: SOAP request sent by Traffic Division

The Traffic Division sends the request message with the parameter specifying the social security number of the citizen investigated to the Population Office server. This request message can be observed in Figure 5.6. As soon as the Population Office server receives this request message, it conducts the necessary operations to find the requested citizen record with the given social security number. After retrieval, the record is put into a *SOAP envelope* and sent back as a response message to the requestor, which in this case is the Traffic Division. In this way, the Traffic Division obtains the citizens Population Office record from the *SOAP message body*. It should be noted that the Population Office could offer the same service in a way that it merely provides the address of the citizen. The SOAP message returned is presented in Figure 5.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="PopRec.xslt"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body>
      <GetRecordR
         <Name>
            <LastN           /LastName>
            <FirstName>Ayisigi Basak</FirstName>
         </Name>
         <FathersNa
         <MothersName>Bahar</MothersName>
         <PlaceofBirth>Istanbul</PlaceofBirth>
         <DateofBirth>1981-01-01</DateofBirth>
         <SocialSecurityNumber>1234567890</SocialSecurityNumber>
         <MaritalStatus Single="true"/>
         <RegisteredTo>
            <City>Bursa</C
            <County>Osmangazi</County>
            <Village>Cirishane</Village>
            <EditionNo>020/04</EditionNo>
            <FamilySerialNo>280</FamilySerialNo>
            <SerialNo>22</SerialNo>
         </RegisteredTo>
         <PopulationOfficeI
            <OfficeName>Yenimahalle</OfficeName>
            <RegistryNo>12225</RegistryNo>
            <RegistryDate>1997-09-04</Registr
         </PopulationOfficeInfo>
         <Address>
            <Line1>Me               /Line1>
            <Line2>Sardunya A/2</Line2>
            <County>Cayyolu</County>
            <City>Ankara</City>
            <PostalCode>06810</Pos     de>
         </Address>
      </GetRecordRespons
   </soap:Body>
</soap:Envelope>
```

Figure 5.7 : SOAP response sent by the Population Office

This interaction between the Traffic Division and the Population Office is merely an example. It can be applied to all the other interactions between departments and organizations. It should be noted that in scenarios 2 and 4, the Traffic Division interacts with the insurance company. For example, the Insurance company reports the payment of a renewal fee to the Traffic Division in scenario 4. In this scenario, the Traffic Division would provide a service to insurance companies for this purpose. Departments providing such services can utilize WSDL documents to describe these services.

A central UDDI within the government can be established to keep these WSDL documents and enable facilitated discovery of services that different departments provide. Such a UDDI example has not been implemented in this research due to the fact that it would not provide additional insight. There are already implemented universal business registries (UBRs) that provide such insight and understanding of the UDDI concept. These have been referenced in Chapter 3 when UDDI was discussed in detail.

5.6.4 Providing Layout

In addition to providing structure, XML can provide layout or formatting. As an example, we have provided layout for an XML document containing Population Office records. Here, we have also utilized the Extensible Stylesheet Language Transformations (XSLT). By using XSLT we have provided a table view of the Population Office Records. Other views can be obtained by applying different stylesheets to the XML document at hand. Figure 5.7 displays the table view provided with the application of the XSLT stylesheet `PopRec.xslt` to the `PopRecords.xml` document. An XML-enabled Web browser was used to apply the specify stylesheet. It should be noted that the latest versions of most Web browsers provide this capability.

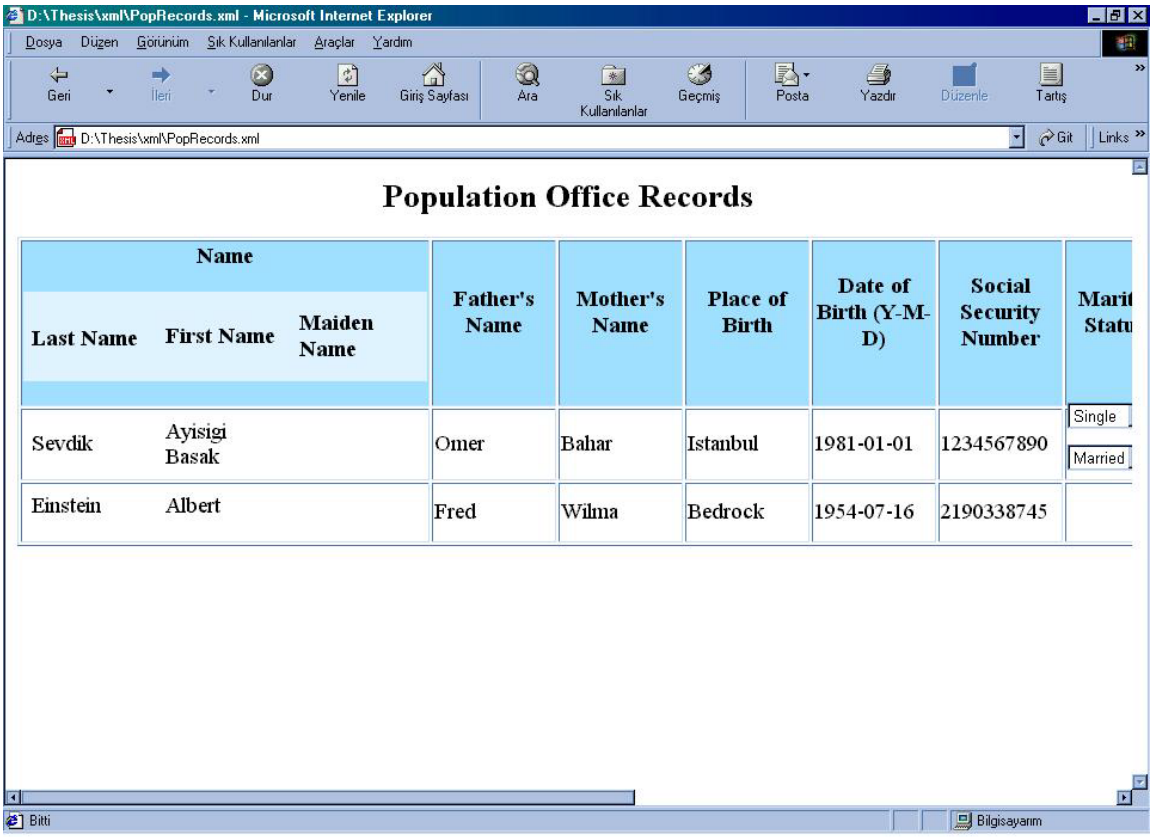


Figure 5.8: Population Office records viewed after XSLT formatting

Throughout the examples in this thesis, the trial version of the commercial application *XML SPY* was used for schema preparation, and document creation. There are many application environments that can be utilized developed for this purpose. Hence, utilization of such a tool is not a necessity as XML can simply be coded as text. Such tools are only used to facilitate the code writing task.

Chapter 6
Conclusions and Future Work

Extensible Markup Language, as the flexible, nonproprietary set of standards for tagging information that it is, has displayed that it really has a potential, through the broad interest and acceptance it gained over the past few years in the business community especially for providing Web services. In this thesis, motivated by this potential and the growing efforts towards transforming the traditional government into an electronic government, we have investigated XML's use in electronic government. Thus, we have presented some exemplary scenarios for the purpose of exploring how XML can be made use of in the electronic government environment. To implement these scenarios, we have investigated the usage of XML and its related standards, including SOAP, WSDL and UDDI.

In this study, we identify the benefits that can be gained from utilizing XML in electronic government, as well as the shortcomings of XML and the precautions measures that need to be taken in order to satisfy the requirements of e-government.

The investigation of XML use in the exemplary scenarios considered show that XML adoption would bring its benefits to the electronic government environment. The major potential contributions of XML adoption by the e-government can be listed as follows:

- facilitating discovery, exchange, and processing of information dispersed among different governmental organizations,
- facilitating citizen and business interactions with the government; thus, increasing public participation,
- removing the requirement of customization and new infrastructure in building electronic government services, which may otherwise be necessary,
- increasing possibility of widespread adoption and broad implementation of consistent tagging of data,
- enabling the availability of government services through a variety of devices,
- prevention of recursion of work, and
- increasing efficiency and productivity of the government.

On the other hand, some possible dangers and shortcomings that XML use would bring are assessed to be:

- the fact that certain parts of XML standards are currently not mature enough for government-wide adoption,
- the risk that redundant data definitions will contradict,
- the potential for building proprietary extensions, and
- the need to maintain adequate security.

The conclusion obtained from all these assessments is that; although currently there are some shortcomings of XML technology, these could be compensated for by careful planning and the consideration of some precautions, and overall, XML adoption by the electronic government is accepted to be beneficiary as it enables the provision of e-government goals.

Further research possibilities include exploring the application of XML and related Web standards to the implementation of the security concerns of electronic government scenarios, and the investigation of architecture planning phase for electronic government XML adoption.

References

[1] United Nations Division for Public Economics and Public Administration (UN-DPEPA) and American Society for Public Administration (ASPA). *Benchmarking E-government: A Global Perspective – Assessing the Progress of the UN Member States*. May 2002.

[2] "Türkiye Bilişim Şurası e-Devlet Çalışma Grubu Raporu" (Turkish Informatics Council e-Government Working Group Report), May 10-12th 2002; Ankara, Turkey.

[3] Doug Tidwell. *Web Services – The Web's Next Revolution.* IBM, developerWorks; https://www6.software.ibm.com/developerworks/education/wsbasics/wsbasics-a4.pdf

[4] W3C Communications Team, Bert Bos. *W3C XML in 10 Points.* Revised November 13th 2001. http://www.w3.org/XML/1999/XML-in-10-points.html

[5] Tim Bray, Dave Hollander, Andrew Layman. *Namespaces in XML W3C Recommendation.* January 14th, 1999; http://www.w3.org/TR/REC-xml-names

[6] David C. Fallside. *XML Schema W3C Recommendation.* May 2nd 2001; http://www.w3.org/TR/xmlschema-0/

[7] W3C. *What is XSL?*. http://www.w3.org/Style/XSL/WhatIsXSL.html

[8] Henry Thompson. *The Extensible Stylesheet Language Family (XSL).* W3C. http://www.w3.org/Style/XSL/

[9] James Clark. *XSL Transformations (XSLT) Version 1.0 W3C Recommendation.* November 16th 1999. http://www.w3.org/TR/xslt

[10] Aaron Skonnard, Martin Gudgin. *Essential XML Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More.* Boston, Addison-Wesley, Chp.5; 2002.

[11] James Clark. *XML Path Language (XPath) Version 1.0, W3C Recommendation*. November 16th, 1999. http://www.w3.org/TR/xpath

[12] Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduard Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, Steve Zilles. *Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation*. October 15th 2001. http://www.w3.org/TR/xsl/

[13] Steven DeRose, Eve Maler, David Orchad. *XML Linking Language (XLink) Version 1.0, W3C Recommendation*. June 27th 2001. http://www.w3.org/TR/xlink/

[14] Steve DeRose, Ron Daniel Jr., Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh. *XML Pointer Language (XPointer), W3C Working Draft*. August 16th 2002. http://www.w3.org/TR/xptr/

[15] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh. *XPointer Framework, W3C Recommendation*. March 25th 2003. http://www.w3.org/TR/xptr-framework/

[16] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh. *XPointer element() Scheme, W3C Recommendation*. March 25th 2003. http://www.w3.org/TR/xptr-element/

[17] Steven DeRose, Eve Maler, Ron Daniel Jr., Jonathan Marsh. *XPointer xmlns() Scheme, W3C Recommendation*. March 25th 2003. http://www.w3.org/TR/xptr-xmlns/

[18] Steven DeRose, Eve Maler, Ron Daniel Jr.. *XPointer xptr() Scheme, W3C Working Draft*. December 19th 2002. http://www.w3.org/TR/xptr-xpointer/

[19] NIST. *XML Technologies, NIST Roadmap.* http://xw2k.sdct.itl.nist.gov/Pham/gsa-prod/xmltechnologies.asp

[20] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen. *SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation*. 24 June 2003; http://www.w3.org/TR/SOAP/

[21] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, W3C Web Services Architecture Group. *SOAP Version 1.2 Part 1: Messaging Framework, Editors Copy*. 2003/06/11; http://www.w3.org/2000/xp/Group/2/06/LC/soap12-part1.html

[22] Nilo Mitra. *W3C Web services Architecture group; SOAP 1.2 Implementation Summary.* http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html

[23] OASIS UDDI Spec TC & UDDI.org Working Group. *UDDI Version 2.04 API Specification, UDDI Committee Specification.* July 19th, 2002; http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-200207 19.htm

[24] OASIS UDDI Spec TC & UDDI.org Working Group. *UDDI Version 3.0.1, UDDI Spec Technical Committee Specification.* October 14th, 2003; http://uddi.org/pubs/uddi-v3.0.1-20031014.htm

[25] Butler Group. *The Importance of UDDI.* OpinionWire, April 17th 2003; http://www.serverworldmagazine.com/opinionw/2003/04/17_uddi/shtml

[26] Giga Information Group, *Giga Survey: XML Achieving Mainstream Usage.* April 30th , 2001.

[27] United States General Accounting Office (GAO); *Electronic Government: Challenges to the Effective Adoption of Extensible Markup Language.* June 2002.

[28] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, Jeffrey Schlimmer, Sanjiva Weerawarana, W3C Web Services Description Working Group. *Web Services Description Language Version 2.0 Part 1: Core Language, W3C Working Draft.* November 10th 2003; `http://www.w3.org/TR/wsdl20`